NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

AN ENVIRONMENT CONDUCIVE TO SOFTWARE DESIGN: JMSS SYSTEM FOR MODELING OF SOFTWARE PROCESSES AND PROGRAMS WITH J-MAPS

D. J. Eddy

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

December 1988

ISBN 0-315-49038-1

ABSTRACT

AN ENVIRONMENT CONDUCIVE TO SOFTWARE DESIGN: JMSS SYSTEM FOR MODELING OF SOFTWARE PROCESSES AND PROGRAMS WITH J-MAPS

D. J. Eddy

An environment conducive to software design is proposed, described and implemented. This environment provides a framework to build and store information in a highly organized way. All information about a program is retained for reference and review.

The framework is the database of information about the system and within that framework are building blocks of tables ordered in a specific way. These tables go beyond conventional decision tables to a structured method of approach to programming. A notation used for these tables is called J-Maps. The J-Map notation is based on sets, defined roles of sets and set members and associations between them. The notation supports control flow charts, data flow diagrams, structured pseudo descriptions as well as other popular structured software development techniques, including graphical representation. All information about the program and including the program is stored in a Relational Database providing the designer with the ability to review and study his designs as well as communicate to others his ideas. The system model is based on the software

life cycle.

The goal of this thesis is to describe a method of problem solving, taking into account the software life cycle, that supports a good structure. The strength of the method described lies not only in the development process but in the structure that permits analytical review at the implementation process. Automated analysis for completeness, consistency and redundancy is demonstrated as well as database techniques used to review data flow and control flow.

ACKNOWLEDGEMENTS

Much of the material presented here was developed and discussed over the past several years with Dr. W.M. Jaworski and fellow students. Dr. Jaworski's support is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

"As the complexity and size of programs increase, the programmer is challenged with the task of organizing his program in a manner which will enhance intellectual manageability. Thus, the structure and style are critical in regards to writing programs and verifying their correctness. In recent years, considerable emphasis has been placed on the correctness of programs and techniques for engineering them to be correct. However, more emphasis should be placed on designing languages which facilitate constructing correct programs"[1]. Hughes went on to describe a language ASPL (A Structured Programming Language) that permits the use of decision tables for expressing complex logic.

Programmer productivity, fast prototyping, computer generated code, computer assisted verification, and provably correct software are just some of the topics under discussion in progressive organizations. To become more productive and remain competitive, optimization of methods and procedures requires not only organization but automation when possible.

Organization at the design and specification levels can be assisted by a consistent computer supported design methodology. Automated tools capable of verification, analysis and report writing provide a support environment

for the life cycle of the software. Dependency on selected staff (ie, the individual(s) who developed the software) will not be as great particularly during the maintenance cycle.

To provide a designer with a productive environment requires integration of the many levels of program development. These levels include:

1. Identification of needs
2. Requirement specification
3. Design specifications
4. Software specifications
5. Software implementation
6. Program documentation
7. Testing and operation
8. Program analysis
9. Program modification
10. Program maintenance

One of the more important factors of this environment is that action at one level of program development is not destructive to the other levels. That is, program modification is integrated into the program specification in such a way that the documentation is not out of sync with the existing program even after modification.

The objective of this thesis is to demonstrate a method of reducing errors at all levels of the software life cycle. This will be demonstrated by the organization, consistency, and support system for J-Map constructs. Large systems often become difficult to maintain due to the loss of the relationships between the specification levels (conceptual) and the implementation levels. The development of a system requires many decisions made at many levels in the life

cycle. To maintain a consistent design development requires review and sometimes revision of design strategies applied. Control of these changes requires updating previously made decisions to reflect the new strategy.

The current method used for storage of programs is usually a set of lines, ordered in a specific way with a notation for producing repetitive, conditional and sequential action. Analyzing data flow, control flow or logic flow from conventional programming language code is nearly impossible and at best time consuming. The structure and style of most programming languages, unfortunately, precludes the ability to develop easy to read and understandable programs. Verification of the correctness of the conventional program from a logical perspective as well as at the implementation level leaves a great deal to be desired.

This thesis will demonstrate a method for designing, communicating and implementing software. This strategy will include:

1. A structure that can be analyzed for logic flow, control flow and data flow;
2. A structure that can be verified for correctness using computer assisted checking;
3. A structure that incorporates documentation at all levels of software development cycle;

2.0 ELEMENTS OF BETTER DESIGN SPECIFICATIONS

"One of the great communication tragedies is to watch an organization go through careful planning exercise, step by step, complete with charts and graphs and then turn the strategy over to the 'creatives' for execution. They, in turn, apply their skills and the strategy disappears in a cloud of technique, never to be recognized again."[2]

Considerable effort is required for developing an environment conducive to software design. The time and effort required for software design must be minimized with end results maximized in terms of quality and integrity of the software. The long practiced engineering and architectural methodology of blueprints and design contrasts with the current practice of programmers who attempt to write a program without a comprehensive design. Equally important, and generally lacking, is an understandable, concise design document.

The development of specialized tools for the designer is essential in whatever trade the designer is in. The quality of these tools is a combination of good materials to begin with, and the experience, talent and ability of the tool designer.

The materials utilized for this thesis are the J-Map structures, relational database[3] and the dBASE III Plus[4] language for implementation. These relatively modern

elements are combined to form the support environment for software design.

2.1 SPECIFICATIONS

Programmer productivity is one of the major factors leading to the development of tools for designing and developing programs. In fact, programmer productivity tends to be a much discussed issue in many computer centers.

Project planning includes evaluation of user needs, projected end result, cost and time frame for completion. There are numerous methods used for project planning. However, most seem to suit the analyst's needs rather than the end user. Many projects fail because the end result is not what was expected or it took too long to deliver.

The design and organization of problem specifications used at all levels of program development could provide clients as well as the designer a productive environment for communication. At the highest level of specifications, the clients requirements could be clearly defined. The specification could then be refined to greater detail by the analyst.

Box 1 lists the properties desirable in a specification as described by Hamilton and Zeldin referred to by Martin[5]. These properties, while in principle ideal, in practice vary greatly in difficulty of application. Problem solving, in general, requires a grasp of the problem before it can be

solved correctly.

BOX 1 DESIRABLE PROPERTIES OF A SPECIFICATION

A PROPER SPECIFICATION SHOULD:

Be free from errors
Have conceptual clarity
Be easy to understand by managers, analysts, or programmers
Be presentable in varying degrees of detail
Be easy to create
Be computable (i.e., have enough precision that program code can be generated automatically)
Be formal input to a program code generator
Be easy to change
Be complete
Be traceable when changes are introduced
Be independent of hardware
Employ a data dictionary
Employ a data model based on formal data analysis
Employ a program module library with automatic verification of interface correctness
Employ computerized tools which make it easy to manipulate and change

2.2 DESIRABLE PROPERTIES OF A SPECIFICATION LANGUAGE

Effective communication between the designer, implementor, administrator and end user is a complex scenario at best. The user department specifies the problem in one language while the designer specifies it in another. By the time the specifications are drawn up, the implementor defines the missing details himself. To design a method for the specification of a project, easily understood at all levels, would be one of the main objectives of a specification language.

A list of desirable properties for a specification language as suggested by Martin[6] is given in Box 2.

BOX 2 DESIRABLE PROPERTIES OF A SPECIFICATION LANGUAGE

It should provide a way to think about systems which improves conceptual clarity.

It should be easy to learn and use. At its higher levels it should be usable by non-DP personnel.

It should be computable and program code should be generatable from it automatically.

It should be designed for maximum automation of systems analysis, design, and programming.

It should be rigorous and mathematically based so that its designs are built from provably correct constructs.

Its mathematical basis should be hidden from the average user because most users are terrified of mathematics.

It should be versatile enough to remove the need for all manual use of programming languages. (Manual programming immediately violates the requirement of provable correctness.)

It should extend from the highest-level conceptualization of systems down to the creation of enough detail for program generation. In other words, one language should suffice for complete system creation. The more detailed versions of a specification should be a natural extension of the more general ones. The high-level specifier should be able to decide into how much detail he wants to go before handing over to the implementor.

It should be a common communication medium among managers, designers, implementors, verifiers, maintainers, and documenters.

It should use graphical techniques which are easy to draw and remember.

It should employ a user-friendly computerized graphics tool for building, changing, and inspecting the design. The language should be formal input to automated design.

It should employ testing tools which assist in verification and permit simulation of missing modules so that partially complete designs can be tested.

It should be usable with top-down or bottom-up design and it should integrate these. Most complex systems come into existence through a combination of top-down and bottom-up design. The technique should allow certain elements of a system to be specified in detail while others, possibly parents or ancestors in the hierarchy, are not yet defined.

It should indicate when a specification is complete.

A specification language is a tool to assist in the design and development of quality programs. This does not mean that bad design is going to be improved with a specification. The garbage-in, garbage-out adage prevails.

2.3 PROPERTIES OF POPULAR METHODOLOGIES

There are currently numerous methodologies supporting software design and development. This section reviews briefly the basic features of some of these methodologies.

2.3.1 GANE AND SARSON[7]

The Gane and Sarson approach to system development uses "logical data flow diagrams" as the supporting tool. Using the top down approach, a series of process boxes outline the features of the system. From there, each process box can be "exploded" into a lower-level, more detailed data flow diagram. Repeating this process several times results in an implementation level of diagrams.

The development process used by Gane and Sarson is as follows:

1. Draw a logical data flow diagram
2. Put the detail in a data dictionary
3. Define the logic of the processes
4. Define the data stores
5. Create a functional specification

The data flow diagrams use various symbol conventions to indicate different types of entities. All entities are defined in a data dictionary. Data flow is symbolized by an arrow indicating the direction of flow. Functions on the entities are then defined. The result is a rather large hand drawn graphical representation for relatively small procedures.

2.3.2 YOURDON AND CONSTANTINE[8]

Yourdon and Constantine give considerable descriptions of complexity, modularity, and heuristic development. The development process for data flow diagrams is also explained. They approach system design from the perspective of the input stream eventually producing the desired output stream (and\or vice versa). The system is functionally decomposed with respect to data flow. Each block of the structure chart is obtained by successive application of the engineering definition of a black box that transforms an input data stream into an output data stream. Control flow is eventually overlaid on the structure (implied by a series of lines between the data boxes). The continued factoring process produces a highly detailed definition of the problem. When these transforms are implemented, the result is like an assembly line that merges streams of input parts and output streams of final products.

Figure 2.1 is an example of the method used by Yourdon

and Constantine in their data flow graph and a structure chart.

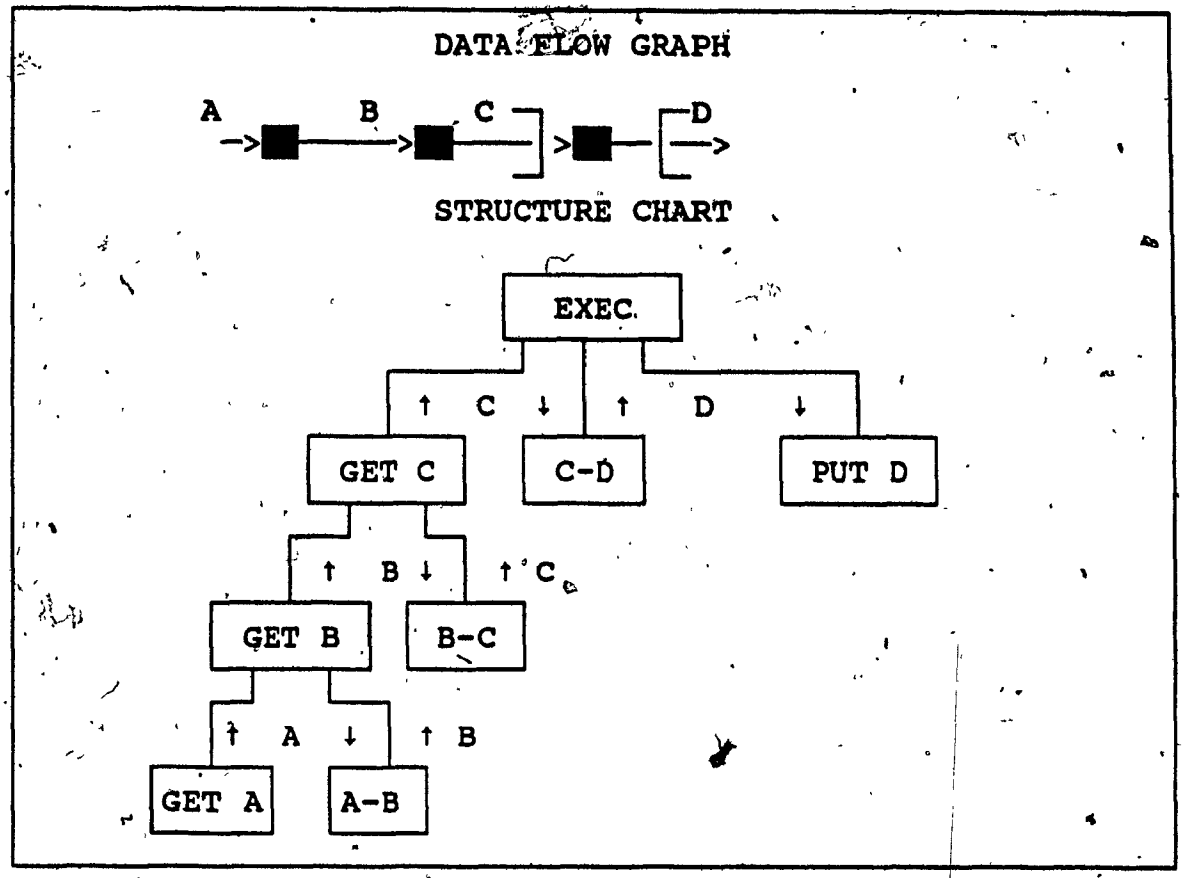


Figure 2.1 Data Flow Design Method

2.3.3 MARTIN[9]

The HOS (Higher-Order Software) methodology described by Martin represents systems by means of trees. Complex systems are decomposed hierarchically in a mathematically rigorous manner. Working out which control structures are needed in the decomposition of a complex operation is tedious. It is regarded as a system

specification tool. The control structures of JOIN, INCLUDE, OR, COJOIN, COINCLUDE, COOR, and CONCUR, based on mathematically provable logic, are used to build the trees. One of the strengths of the system is its ability to generate code from the specifications. It can operate on any computer in any language for which a generator module exists.

2.3.5 JACKSON[10]

The Jackson model of software engineering structures a program in pure tree structures. These structures give rise to the concept of hierarchical modularity.

The programming process can be partitioned into the following steps:

1. Form a system network diagram that models the problem environment.
2. Define and verify the data-stream structures.
3. Derive and verify the program structures.
4. Derive and allocate the elementary operations.
5. Write the structure text and program text.

Jackson partitions complex problems into simple programs resulting in a network of hierarchies. The data structure design methodology of Jackson is developed from the bottom up. The "correct" method for extending this methodology to large system problems is still being developed.

An example of Jackson's data structure design is given in Figure 2.2 as given in Bergland's review[11].

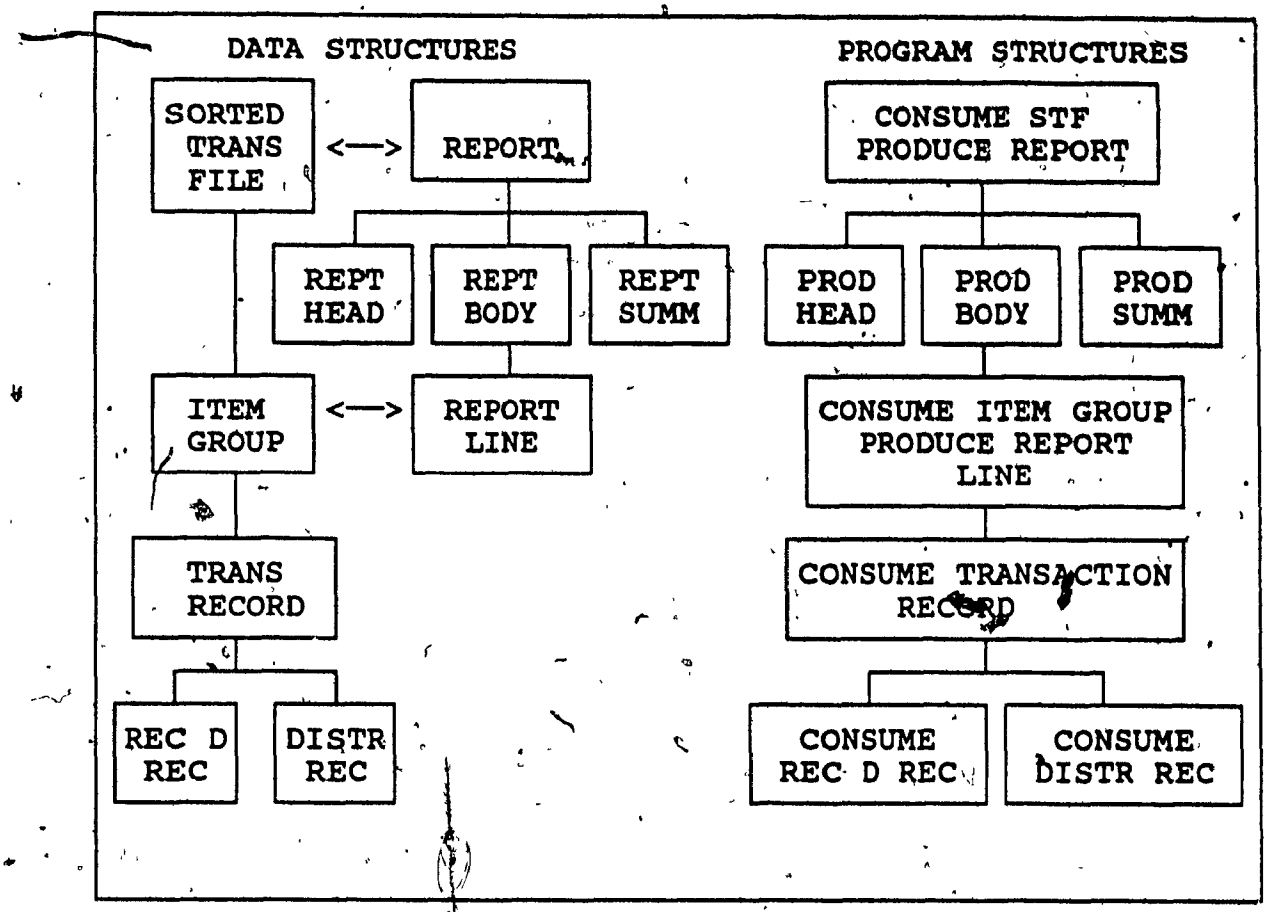


Figure 2.2 Data and Program Structure used by Jackson

This data structure (Figure 2.2) is then overlayed with executable operations. An example of this overlaying process is given in Figure 2.3.

The "Text Step" translates the structure diagram into structure text as shown in Figure 2.4. Program labels tend to be limited in size and consequently require abbreviations of words. This translation defines how the program should be coded.

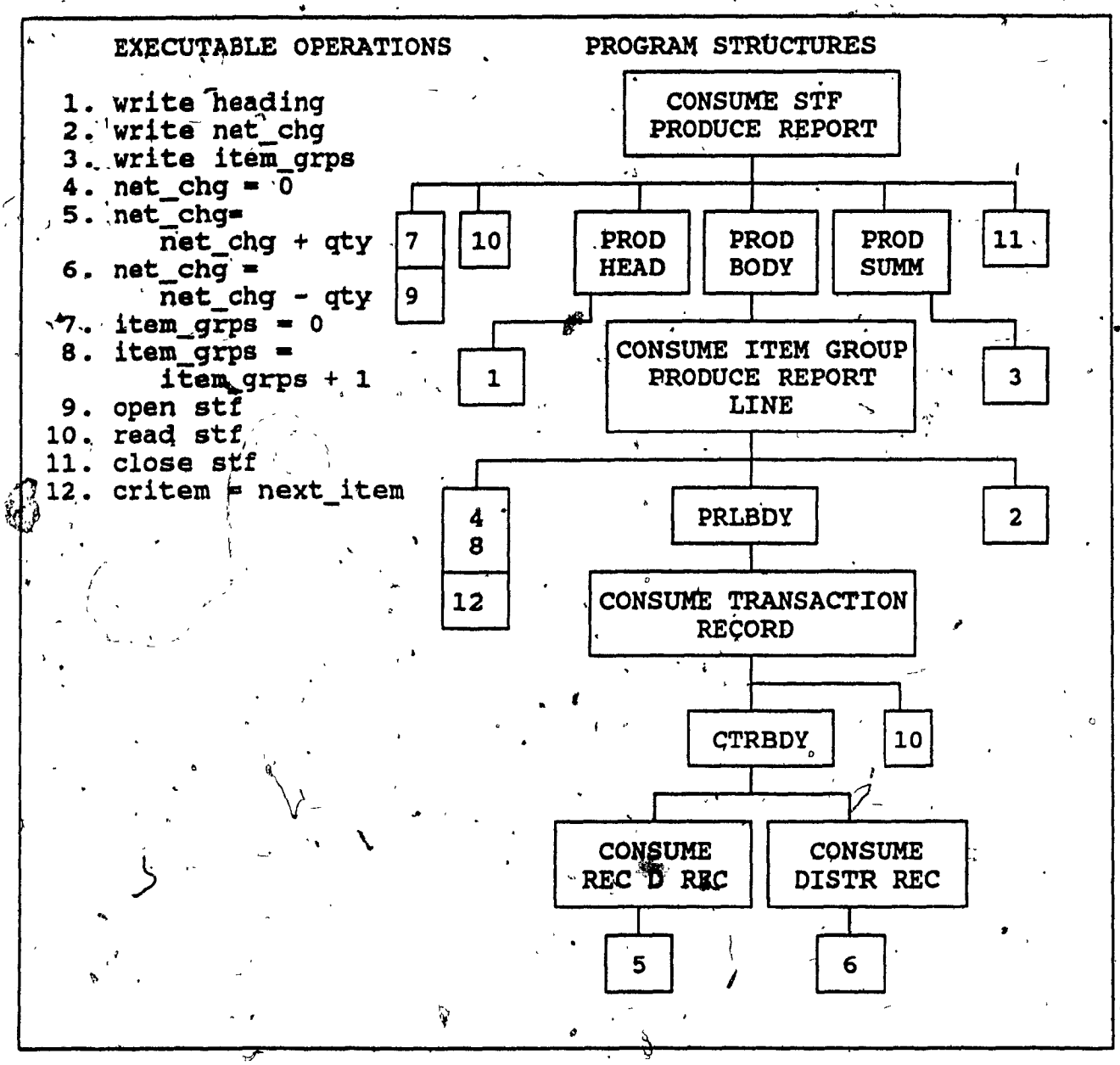


Figure 2.3 Allocation of Executable Operations to Algorithm in Figure 2.2 (Jackson)

```
P seq
   item_grps:= 0;
   open stf; read stf;
   write heading;
   P_BODY itr until (eof_stf)
        C_ITEM_GRP_P_REPT_LINE seq
             net_chg = 0;
             item_grps:=item_grps + 1;
             critem:= next item;
             PRLBDY itr while (next item = critem)
                  C_TRANS_REC seq
                       CTRBDY sel (code - R)
                       net_chg:= net_chg + qty;
                       CTRBDY alt (code - D)
                       net_chg:= net_chg - qty;
                       CTRBDY end
                       read stf;
                  C_TRANS_REC end
             PRLBDY END
             write net_chg;
        C_ITEM_GRP_P_REPT_LINE end
   P_BODY end
   write item_grps; close stf;
P end
```

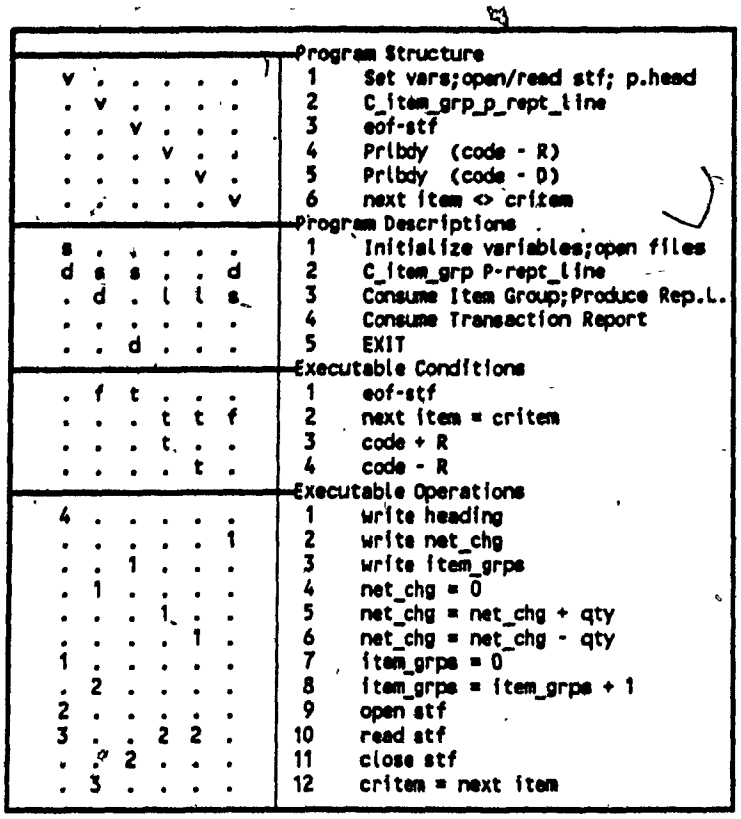
Figure 2.4 Structure text for Figure 2.2 (Jackson)

2.3.6 JAWORSKI[12,13]

J-Maps* are related to decision tables[14] to a limited extent. Decision tables are principally truth tables. This allows the evaluation for a complete set of alternatives by simple and direct means. Computer assisted evaluations for completeness are quite straight forward. Decision tables

---

* J-Maps have evolved over the past 12 years from a concept developed by Dr. W.M. Jaworski. Previous names given to this type of structure were "ABL" and "W4". All (ABL, W4 and J-Maps) differ somewhat in structure and concept.



							Program Structure
v	.	.	.	.	.	1	Set vars;open/read stf; p.head
.	v	.	.	.	.	2	C_item_grp_p_rept_line
.	.	v	.	.	.	3	eof-stf
.	.	.	v	.	.	4	Pribdy (code - R)
.	.	.	.	v	.	5	Pribdy (code - D)
.	.	.	.	.	v	6	next item <> critem
							Program Descriptions
s	.	.	.	.	.	1	Initialize variables;open files
d	s	s	.	.	d	2	C_item_grp P-rept_line
.	d	.	l	l	s	3	Consume Item Group;Produce Rep.L.
.	.	.	.	.	.	4	Consume Transaction Report
.	.	d	.	.	.	5	EXIT
							Executable Conditions
.	f	t	.	.	.	1	eof-stf
.	.	.	t	t	f	2	next item = critem
.	.	.	t	.	.	3	code = R
.	.	.	.	t	.	4	code - R
							Executable Operations
4	.	.	.	.	.	1	write heading
.	.	.	.	.	1	2	write net_chg
.	.	1	.	.	.	3	write item_grps
.	1	.	.	.	.	4	net_chg = 0
.	.	.	1	.	.	5	net_chg = net_chg + qty
.	.	.	.	1	.	6	net_chg = net_chg - qty
1	.	.	.	.	.	7	item_grps = 0
.	2	.	.	.	.	8	item_grps = item_grps + 1
2	.	.	.	.	.	9	open stf
3	.	.	2	2	.	10	read stf
.	.	2	.	.	.	11	close stf
.	3	.	.	.	.	12	critem = next item

Figure 2.5 J-Map Table Representation of the Algorithm in Figure 2.2

are very limited in that they are able to evaluate one set of alternatives. There is no control flow defined to other tables. J-Maps combine the features of a decision table with other important concepts. J-Maps are composed of one or more decision tables linked to form a unit with a defined control flow.

Documentation is imbedded in the J-Map tables keeping

Documentation is imbedded in the J-Map tables keeping human logic mapped on machine logic at each decision point (cluster) and each selection item (alternative). The usual development process of the J-Maps is a top down approach of functional decomposition (though bottom up can also be used). Decomposition may be performed with respect to time order, data flow, logical grouping, access to a common resource, control flow or some other criterion. The result is a flexible environment capable of supporting many different philosophies of how to design and develop programs.

Figure 2.5 represents the algorithm in Figure 2.2 in J-Map format. The program structure is defined by the relationships between the structure and the program description. The first alternative in the program structure is indicated by a "v" in row 1, column 1. It states: 1. Set variables; open sorted transaction file; read sorted transaction file; print heading of report. Looking down the first column an "s" indicates the starting cluster, in this case 1) Initialize variables; Open files. There are no conditions guarding this alternative, so the actions list is indicated by numbers in the first column. They are:

1. item_grps = 0
2. open stf
3. read stf
4. write heading

		Program Structure
v	. . . . .	1 Set vars;open/read stf; p.head
		Program Descriptions
s	. . . . .	1 Initialize variables;open files
d	. . . . .	2 C_item_grp P-rept_line
		Executable Operations
4	. . . . .	1 write heading
1	. . . . .	7 item_grps = 0
2	. . . . .	9 open stf
3	. . . . .	10 read stf

Figure 2.6 J-Map Table Showing Cluster 1 in Figure 2.5

After these operations the next step, indicated by the "d" in the first column, shows step 2 as the next step, as shown in Figure 2.7. There are two alternatives to select from, #2 or #3. The description of this cluster is " 2. Consume item group and produce report line". The descriptions of the alternatives are: 2. C_item_grp; P_rept_line or 3: eof-stf (sorted transaction file).

		Program Structure
. v	. . . .	2 C_item_grp_p_rept_line
. .	v	3 eof-stf
		Program Descriptions
. s	s	2 C_item_grp P-rept_line
. d	. . . .	3 Consume Item Group;Produce Rep.L.
. .	d	5 EXIT
		Executable Conditions
. f	t	1 eof-stf
		Executable Operations
. .	1	3 write item_grps
. 1	. . . .	4 net_chg = 0
. 2	. . . .	8 item_grps = item_grps + 1
. .	2	11 close stf
. 3	. . . .	12 crItem = next item

Figure 2.7 J-Map Table Showing Cluster 2 in Figure 2.5

Checking the guards to alternative 2, assume the state "eof-stf" is false. In this case, alternative 2 is selected with the actions 4,8, and 12 completed. The next cluster is number 3. Repeating the process of selection, one alternative from the alternative set <4,5,6> will match the current state of the machine (Figure 2.8). Assume

		Program Structure
. . . v . .	4	Pribdy (code - R)
. . . . d .	5	Pribdy (code - D)
. v	6	next item <> critem
		Program Descriptions
. . . . . d	2	C_item_grp P-rept_line
. . . l l s	3	Consume Item Group;Produce Rep.L.
		Executable Conditions
. . . t t f	2	next item = critem
. . . t . .	3	code + R
. . . . t .	4	code - R
		Executable Operations
. 1	2	write net_chg
. . . 1 . .	5	net_chg = net_chg + qty
. . . . 1 .	6	net_chg = net_chg - qty
. . . 2 2 .	10	read stf

Figure 2.8 J-Map Table Showing Cluster 3 in Figure 2.5

alternative 5 is selected. The action list is completed for that alternative. The next cluster remains number 3 (indicated by the "1"). Assume this time alternative 6 is selected (this means next_item <> critem in the guard). After action #2 is complete the "d" in this column indicates the next cluster as #2. Going back to cluster 2 (see Figure 2.7), assume the current state of the machine is "eof_stf=.T.". Then the program ends after the 2 actions, #3 and #11.

This programming structure will be discussed in greater detail in the following chapters. What should be noted here is the ease with which trees can be represented in table format. This is an important feature in that tables are more easily generated than trees. Also, the ability to review the condition lists and check for completeness is valuable in determining if all possible conditions have been considered.

The development processes used in other methodologies can be applied using the J-Map approach. Report writing and graphical representation of the programs are easily produced when J-Maps are used.

The strength of a methodology is often governed strongly by the practical application of the methodology.

2.4 CONCLUSIONS

G. L. Bergland[15] reviewed several program design methodologies. He stated " The major motivation for looking at program design methodologies is the desire to reduce the cost of producing and maintaining software. ... I believe that the quality of the program structure resulting from a design methodology is the single most important determinant of the life-cycle costs for the resulting software system."

Figure 2.9 compares the design methodologies reviewed. All concentrate on the software specification and software implementation processes in the system life cycle. The

activities used in each methodology varies. All methodologies provide some insight into what the "ideal" methodology should offer.

Model of Program Design Methodologies in J-Map Format		
		CLASSIFICATION OF DESIGN METHODOLOGIES
. . V . .	1	Functional Decomposition
. . V . V	2	Data Flow Design
V . . V .	3	Data Structure Design
. . . . .	4	Programming Calculus[16]
		METHODOLOGY
V . . .	1*	M.A. Jackson
. V . .	2*	E. Yourdon and L.L. Constantine
. . V .	3*	Gane and Sarson
. . . V	4	J. Martin
		SYSTEM LIFE PHASE
. . . .	1	Identification of Needs
. . . .	2	Requirement Specification
. . . .	3	Design Specifications
V V V V	4	Software Specification
V V V V	5	Software Implementation
. . . .	6	Testing
. . . .	7	Operation
. . . .	8	Maintenance
. . . .	9	Retirement
		ACTIVITY
. V . V	1	Data Flow Diagram
. . V V	2	Contents of Data Stores
. . V .	3	Process Logic
. V . .	4	Structure Chart
V . . V	5	Input/Output Data Stream
V . . V	6	Program Structure
V . . V	7	Structure Text - Pseudocode

Figure 2.9 A Comparison of Software Methodologies

* Comparison study taken in part from the Jaworski and Radhakrishnan[17] review.

The next section describes a practical approach to system design using J-Map structures.

3.0 AN ENVIRONMENT DESIGNED FOR SOFTWARE DESIGN

"One indication of the validity of a principle is the vigour and persistence with which it is opposed. In any field, if people see that a principle is obvious nonsense and easy to refute, they tend to ignore it. On the other hand, if the principle is difficult to refute and it causes them to question some of their own basic assumptions with which their names may be identified, they have to go out of their way to find something wrong with it."

Charles Osgood

3.1 WHAT IS IT ?

There are currently many design methodologies to choose from. To compare the attributes and weaknesses of each methodology requires a review of the needs of a system designer.

The stages of system development is a layered process. Figure 3.1 shows one view of this process. The beginning stages (top) are less structured and more conceptual. Conversely, lower stages are very structured. The flow of information between the various levels during system development is extremely useful.

Figure 3.2A represents the Software Life Cycle with a more explicit view of the relationships between "Pre Development Processes", "Development Process" and "Post

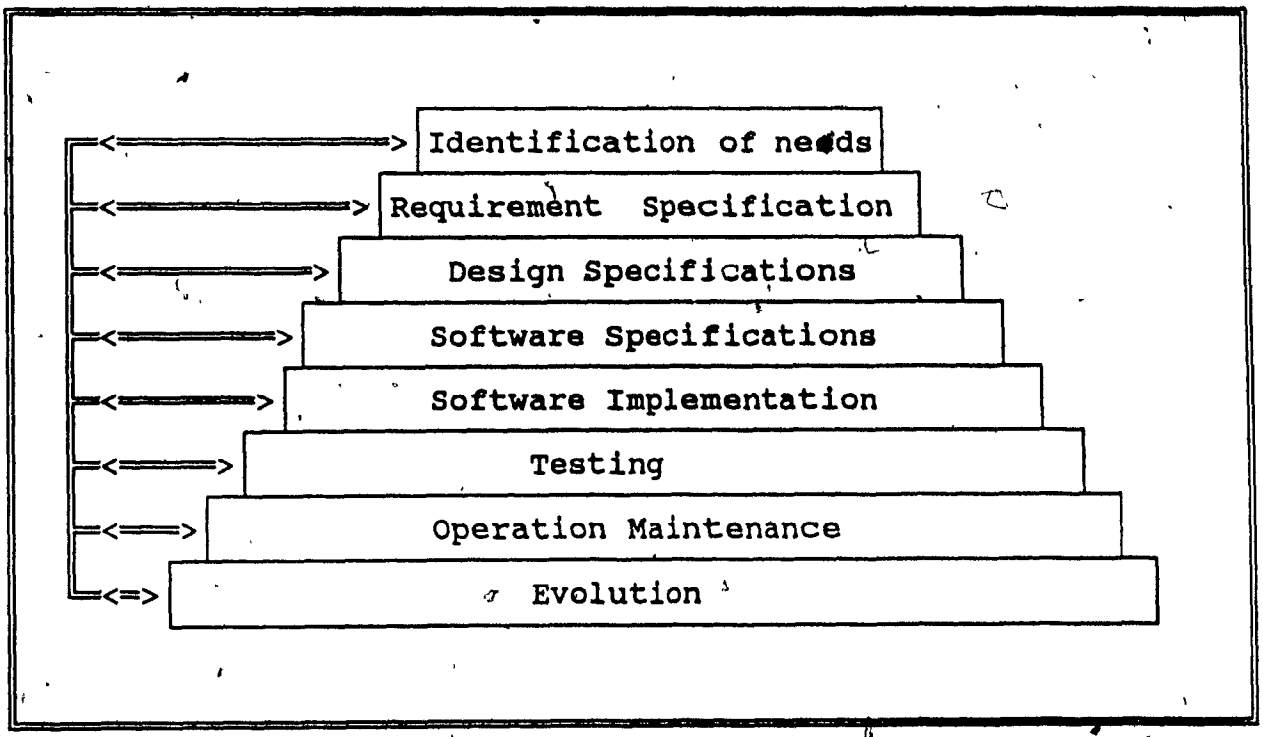


Figure 3.1 Stages of System Development

Development Processes". In this case, the "Pre Development Process" includes "Identification of Needs" or item number 1 in the process list. The "Development Process" includes, items 2,3,4,5 and 6 in the process list. The "Post Development Processes" includes items 7,8, and 9 in the process list. Reading the table in a column wise manner, each step is composed of two alternatives. The first alternative represents looping (1) through a list of activities as many times as required. The second alternative moves to the next expected process. This means action is moved from the source process, "s", to the destination process, "d". However, system design is seldom

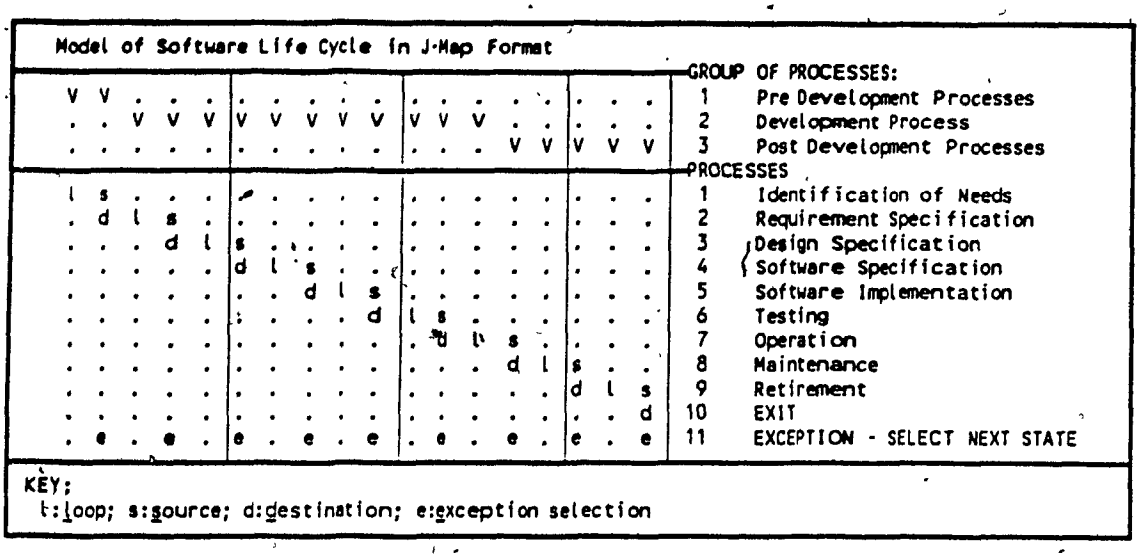


Figure 3.2A Model of a Software Life Cycle in J-Map Format

a planned sequential list of activities. Rather repetition of a previous process or jumping to another process such as testing may be desirable at any given time. This case is described as the exception (e) state, allowing for the non-expected selection of the next process.

The model is expanded further by incorporating the activities for each process. Figure 3.2B shows the the list of activities for each process. These activities were not decomposed according to time, but rather according to the process. Consequently, only "v"s indicate these activities rather than a particular order sequence.

3.2 WHAT DOES IT DO?

The relationships between the levels of definition is shown graphically in Figure 3.3. The horizontal spreading

Model of Software Life Cycle in J-Map Format

																			GROUP OF PROCESSES:
V	V	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	Pre Development Processes
.	.	V	V	V	V	V	V	V	V	V	V	V	.	.	.	.	.	2	Development Process
.	.	.	.	.	.	.	.	.	.	.	.	.	V	V	V	V	V	3	Post Development Processes
																			PROCESSES
l	s	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	Identification of Needs
.	d	l	s	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2	Requirement Specification
.	.	.	d	l	s	.	.	.	.	.	.	.	.	.	.	.	.	3	Design Specification
.	.	.	.	.	d	l	s	.	.	.	.	.	.	.	.	.	.	4	Software Specification
.	.	.	.	.	.	.	d	l	s	.	.	.	.	.	.	.	.	5	Software Implementation
.	.	.	.	.	.	.	.	.	d	l	s	.	.	.	.	.	.	6	Testing
.	.	.	.	.	.	.	.	.	.	.	d	l	s	.	.	.	.	7	Operation
.	.	.	.	.	.	.	.	.	.	.	.	.	d	l	s	.	.	8	Maintenance
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	d	l	s	9	Retirement
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	d	10	EXIT
.	e	.	e	.	e	.	e	.	e	.	e	.	e	.	e	.	e	11	EXCEPTION - SELECT NEXT STATE
																			ACTIVITY
V	.	V	.	V	.	V	.	V	.	V	.	V	.	V	.	V	.	1.	Program ID
V	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2.	System Requirements
V	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3.	Design Constraints
V	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	4.	User Needs
V	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	5.	Functional/performance requirement
.	.	V	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	6.	Functions (overview)
.	.	V	.	V	.	.	.	.	.	.	.	.	.	.	.	.	.	7.	Performance
.	.	V	.	V	.	.	.	.	.	.	.	V	.	V	.	V	.	8.	External Interfaces
.	.	V	.	V	.	.	.	.	.	.	.	.	.	.	.	.	.	9.	Quality Attributes
.	.	V	.	V	.	.	.	.	.	.	.	.	.	.	.	.	.	10.	Design Constraints
.	.	.	.	V	.	.	.	.	.	.	.	V	.	V	.	V	.	11.	Functions (specific)
.	.	.	.	V	.	.	.	.	.	.	.	V	.	V	.	V	.	12.	Internal interfaces
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	13.	Process
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	14.	Data Store
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	15.	Data Object
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	16.	Data Flow
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	17.	Control Data Flow
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	18.	Entity
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	19.	Binary Relationship
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	20.	Multi-Relationship
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	21.	Attribute
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	22.	Procedure
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	23.	Alternative
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	24.	State
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	25.	Pre-Condition
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	26.	Action
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	27.	Post Condition
.	.	.	.	.	.	V	.	V	.	.	.	.	.	.	.	.	.	28.	Operation
.	.	.	.	.	.	.	.	.	.	V	.	.	.	.	.	.	.	29.	Interpreter
.	.	.	.	.	.	.	.	.	.	V	.	.	.	.	.	.	.	30.	Generator
.	.	.	.	.	.	.	.	.	.	.	.	V	.	V	.	.	.	31.	Changes
.	.	.	.	.	.	.	.	.	.	.	.	V	.	V	.	.	.	32.	Observations
V	.	V	.	V	.	V	.	V	.	V	.	V	.	V	.	.	.	33.	Report Writer

KEY:
l:loop; s:source; d:destination; e:exception selection

Figure 3.2B Model of a Software Life Cycle in J-Map Format

(or number of boxes on the row) represents the size of the problem/solution. The vertical direction represents the different levels of definition of the problem/solution.

At the identification of needs level, conceptual descriptions are given with some definition of major parts of the system. The requirements specification defines more specifically those major divisions as "functions". The design specification level becomes more refined. In the example given (in section 3.3) there may be 3 design specifications, each representing one of the functions defined in the requirements specification. The sample design specification (Figure 3.11), describes the first function in the requirement specification (Figure 3.9), namely, "The data entry/editor to support the J-Map notation." The software specification is a refinement of the design specification. In this example, the software specification (Figure 3.13) defines the first function in the design specification, namely, "The menu driven screens for selection of all activities".

This level then expands broadly into the software implementation definition. In this case there may be a dozen or more menu screens, each representing a procedure in the program. Figure 3.3 graphically shows how this spreading occurs. The horizontal divisions would relate to the size of the program. The more divisions, the larger the program. The vertical expansion indicates the increase in

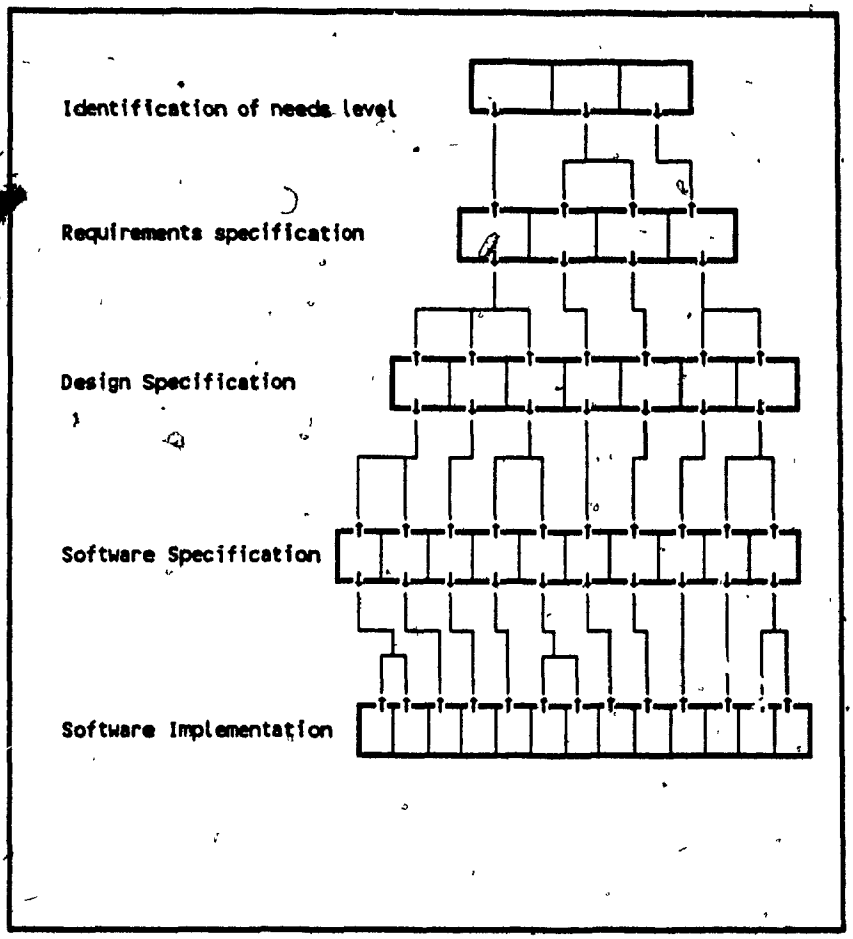


Figure 3.3 Relationships Between Levels of Definition

the amount of information required as program development processes are completed.

This partitioning process keeps highly interrelated parts of the problem in the same piece of the system. From a management perspective, it has been suggested by Yourdon and Constantine that "... implementation, maintenance, and modification would be minimized if the system could be designed in such a way that its pieces were small, easily related to the application, and relatively independent of

one another. This means, then, that good design is an exercise in partitioning and organizing the pieces of a system."[18]

The Software Life Cycle model selected here is appropriate for most organizations. This model becomes the framework for software projects. It is also similar to the model suggested by the IEEE (Preliminary) Report[19].

The management of the volume of information a Software Life Cycle (SLC) can encompass requires considerable organization. More importantly, easy access to the information is of considerable importance in order to remain on tract and in control of the software development. The SLC development process is not a sequential list of activities. Rather, the development activities include many loops and decision points not unlike the program itself.

The amount of information required in the design process of a program expands considerable with each step in the program development process. The details of each level are defined in Figure 3.2B. Each level may not be all-inclusive. However, they offer the designer a good basis for structured design.

J-MAPS expand on the concept of sets and their relationships. The J-MAP model used here is a subset of the components used by many other popular methodologies as outlined in Jaworski and Radhakrishnan[20] and described in section 2.4.

The logical overview of how information can be represented is described in Figure 3.4A. Information can be displayed or viewed in one of five types of perspectives. Sometimes, the same information may be viewed in several different ways. These types of perspective include:

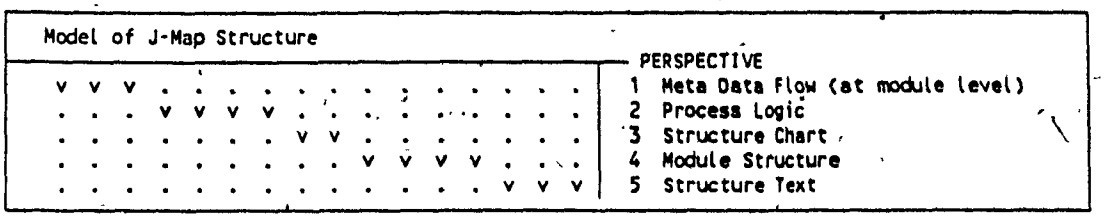


Figure 3.4A Model of the J-Map Structure

These perspectives are useful at different stages of system development. The perspective available for each process is show in Figure 3.4B.

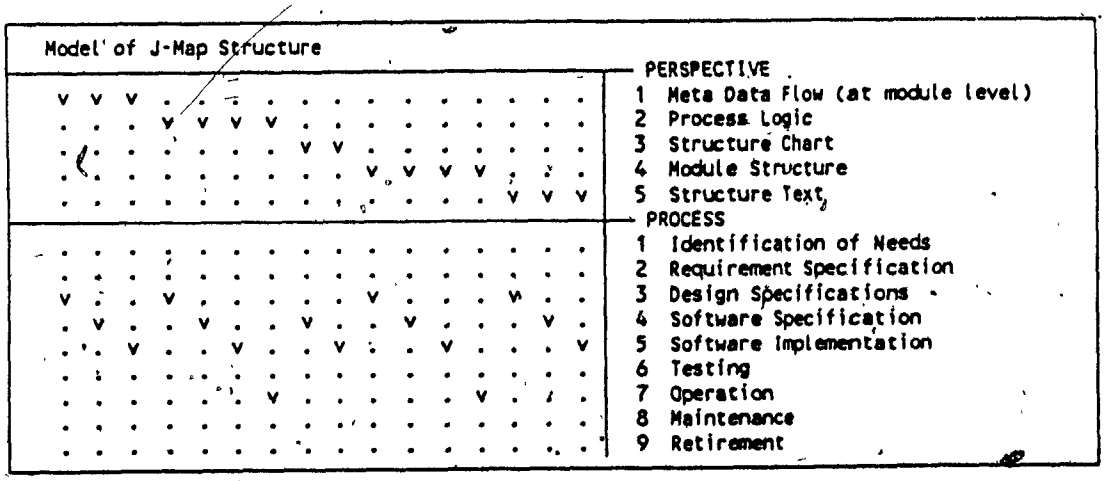


Figure 3.4B Model of the J-Map Structure

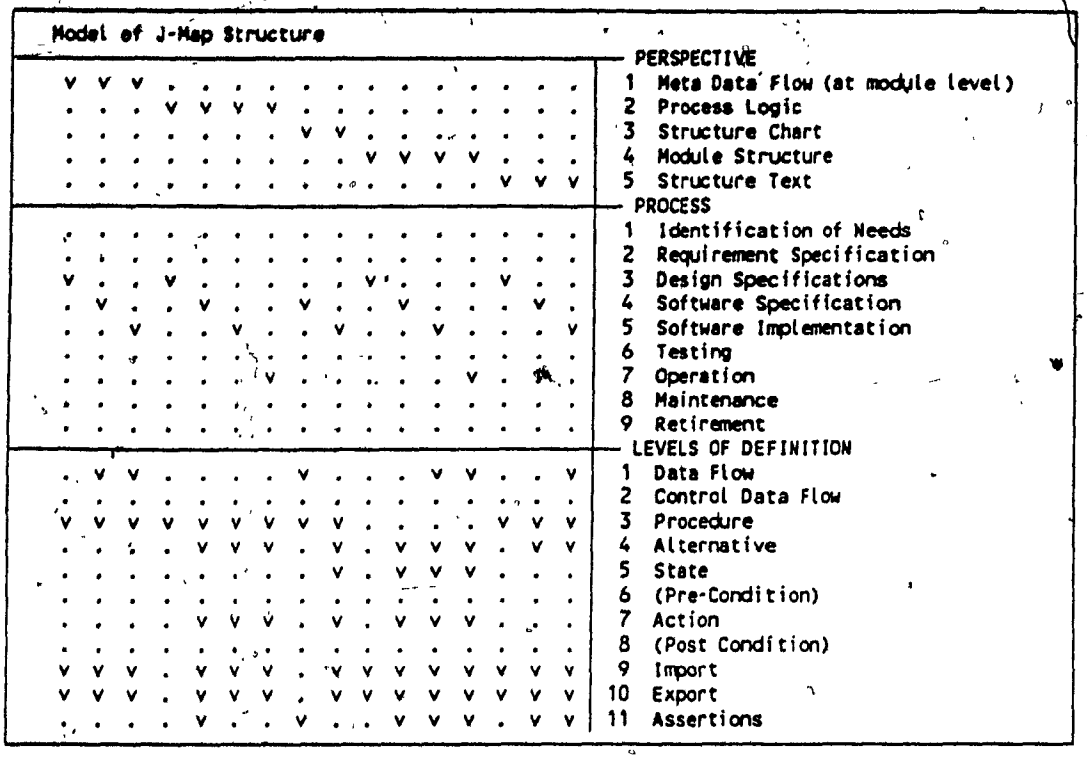


Figure 3.4C Model of the J-Map Structure

Further decomposition gives the levels of definition available to each process. These definitions are shown in Figure 3.4C.

Further decomposition still is shown in Figure 3.4D with element roles for each of these perspectives at different stages of the software life cycle. To read the table, select the item of interest listed under "PERSPECTIVE". Select from that row one of the columns marked with a "v". Now reading column-wise, note the system life phase or stage the column defines. The various types of information defined are listed under "Levels of definition" and finally

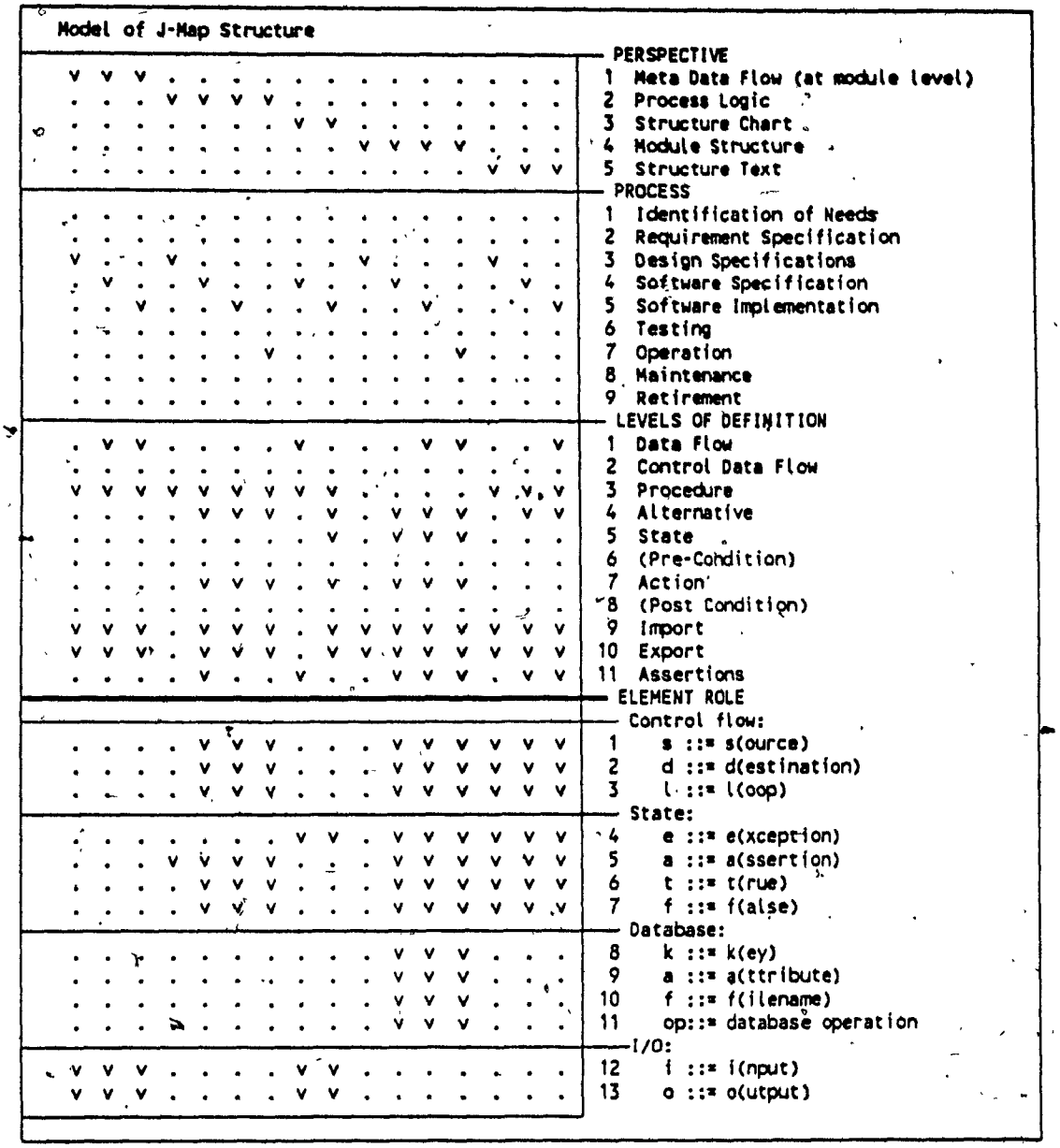


Figure 3.4D Model of the J-Map Structure

the element roles are described.

Take for instance "Module Structure". This structure is useful at four levels of system life phase defined in this model. At the specification level, only the information imported to or exported from the module is defined. At the implementation level, the imported and exported information may be implied or explicit. An example of implied imported information would be using a database, opened prior to calling the module. The module may not define the database, the index file and perhaps even the record pointer position, but will use information found in that record. This is imported implied status information. It may be crucial to the module, but may not be defined in most conventional program documentation. Other implied imported information would include global variables. Explicit imported information would be parameters passed to the module.

At the Design stage, more information is required and defined as shown in the table. Each succeeding phase in the system development process builds on previous defined and stored information.

The method described here demonstrates a technique for partitioning and organizing the pieces of a system. More importantly, this method can store this information in an organized manner during the development of the system. This

linkage between levels becomes very useful as the system develops. Such things as limits set at the requirement specification level will keep the project on course. Functional and performance requirements defined in the identification of needs are easily reviewed and not easily forgotten. Design constraints must always be held in perspective at all levels of development. It should be remembered, however, that some constraints may not be initially apparent. The implementation level often brings some surprises. The ability to modify and update the higher specification levels during the development at lower levels, such as the implementation level, provides the flexibility required in real time programming. Also, this information becomes invaluable during the maintenance and revision levels by preventing repetition of original errors or directions.

3.3 J-MAPS: IMPLEMENTATION LEVEL

J-Map tables allow for the development of control flow in a very structured way. This structure is encompassing enough to eliminate the need for overlapping constructs such as

1. Do While
2. If ... then ... else
3. For
4. Case

just to name a few in standard programming languages. Control flow is controlled by selection of an alternative within a set or cluster. That alternative specifies the conditions required for the alternative, a sequential list of actions, and then what to do next. This structure is easily stored in a database structure allowing for analysis of control flow.

Also stored in the database are the action and condition statements. These statements could be further broken down into elements. These elements, also stored in a database, would allow for analysis of data flow.

Decision tables are enhanced in such a way as to integrate documentation into the programming structure. This way documentation always corresponds to the program even after modification. The documentation is also descriptive at the various levels of program refinement. This means program specification may be made at various levels of refinement. At the highest level of refinement, analyst and client would be able to communicate. At the lowest or finest level of refinement program code can be generated from it automatically.

The J-MAP table combines the decision table structure with several additional important features. These features form the relationships between decision tables and provide program specification. The types of relationships are: sequence, selection, repetition, and nesting. Figure 3.5A

TABLE HEADING								ID	TYPE	DESCRIPTION
v								1	A	
	v	ALTERNATIVE BLOCK						2	A	
		v						3	A	
			v					4	A	
				v				5	A	
					v			6	A	
						v		7	A	
							v	8	A	
s		CLUSTER BLOCK						1	C	
d	1	s	s	d			d	2	C	
		d		s	s	1	s	3	C	
			d		d			4	C	EXIT

Figure 3.5A The J-Map Table Construct

describes some of the features of a J-Map table. Several features have been added to the basic decision table. The "CLUSTER" signifies the decision table blocks. In this case there are THREE decision tables (shown separated by double lines) in this J-Map Table. In the first decision table, there is only 1 rule or alternative. The second decision table contains 3 alternatives, the third 4 alternatives. The last cluster is the exit cluster. The next step is defined in the cluster block with "d" or "1" indicating the next cluster. This provides the relationships between clusters. The selection after cluster 1 (after completion of the actions) is cluster 2 (or table 2). Cluster 2 contains three alternatives. Depending on the rule or alternative that matches the current state, the selection available from cluster 2 is either:

- repetition of table 2
- sequencing to table 3
- exit

The definition of "NESTING" is limited to mean only a call to another J-Map table from an action. This meaning could have extended to apply to a nesting relationship if one table is completely interpreted while testing a condition. This would be appropriate if the language used to support the logic included function statements. However, dBASE does not.

TABLE HEADING								ID	TYPE	DESCRIPTION
v								1	A	
	v	ALTERNATIVE BLOCK						2	A	
		v						3	A	
			v					4	A	
				v				5	A	
					v			6	A	
						v		7	A	
							v	8	A	
s		CLUSTER BLOCK						1	C	
d	l	s	s	d			d	2	C	
		d		s	s	l/	s	3	C	
			d		d			4	C	EXIT
	t	t	f					1	P	
		CONDITION BLOCK				t	f	2	P	
	t	f		t	t	t		3	P	
				t	f			4	P	

Figure 3.5B The J-Map Table Construct

Additional information is added to the table by

introducing the condition block (Figure 3.5B shows this addition). This block defines the control flow of the algorithm. "P" represents the predicate or condition description.

TABLE HEADING								ID	TYPE	DESCRIPTION
v								1	A	
	v	ALTERNATIVE BLOCK						2	A	
		v						3	A	
			v					4	A	
				v				5	A	
					v			6	A	
						v		7	A	
							v	8	A	
s		CLUSTER BLOCK						1	C	
d	1	s	s	d			d	2	C	
		d		s	s	1	s	3	C	
			d		d			4	C	EXIT
	t	t	f					1	P	
		CONDITION BLOCK				t	f	2	P	
	t	f		t	t	t		3	P	
				t	f			4	P	
1								1	X	
2		1	1					2	X	
3								3	X	
	1	2	2					4	X	
	2	ACTION		BLOCK		2		5	X	
			3		2			6	X	
		3						7	X	
				1	1	1	1	8	X	
				2			2	9	X	

Figure 3.5C The J-Map Table Construct

The final block added is the action block (Figure 3.5C). The sequence of actions are indicated by the numbers in that block. The numbers indicate the sequence with which to institute the actions. This completes the standard format

of the J-Map implementation construct.

This map could be refined to the next lower level by further defining some components. This would relate to the top down approach in system design. The J-Map table structure for the implementation phase is described in detail in Chapter 5.

3.4 WORKING THROUGH A PROBLEM

The following example, taken from the J-Map Support System (JMSS) design, gives a demonstration of how the process works. The various examples of "Identification of the Needs", "Requirement Specification", etc., are actual excerpts taken from the JMSS. Refer to Figure 3.2 for the relationships between processes.

3.4.1 STARTING WITH AN IDEA

The idea level for this project started as a list of conceptual stages for the development of a support technology for J-Maps. Figure 3.6 is a list of the starting ideas.

The idea level is a rough stage used to correlate ideas for the development of the information system. By determining the system requirements it can also be determined what will be delivered. Analyzing design constraints and putting a limit on the design is essential long before the implementation level is reached. The

1. Development of a data entry procedure to support the J-Map notation;
2. Verification of data entry;
3. Design a worksheet display on 24x80 character screen(s) in J-Map notation;
4. Allow for flexible selection of items for display;
5. Interpreter;
6. Program generator.

Figure 3.6 Example: Starting with Ideas

1. Determining System Requirements;
2. Documenting users needs;
3. Partitioning and allocating functional and performance requirement to obtain detailed requirements;
4. Analyzing design constraints - putting a limit on design

Figure 3.7 Idea Development Features

```
IDENTIFICATION OF THE NEEDS (IDEAS)
        I.  SYSTEM REQUIREMENTS
v . . .   1.  Develop an data entry procedure to support the J-Map notation.
. v . .   2.  Develop an editor to support the J-Map notation
. . v .   3.  Develop an interpreter
. . . v   4.  Develop a program generator
       II.  USER NEEDS
. v . .   1.  To be able to select specific information for screen display
. v . .   2.  Incorporate structured software development techniques - dataflow, entity-relationship
              diagrams; structural charts, control flow graphs; structured development, walk-throughs
. . v .   3.  Interactive modes available
. . . v   4.  Reasonably good performance in generating programs

      III.  FUNCTIONAL AND PERFORMANCE REQUIREMENTS
v v v v   1.  To be fast enough not to frustrate the user
v v v .   2.  To be flexible
v v v v   3.  To be used as a computer-aided system for rapid prototyping
v v v v   4.  To accommodate new as well as experienced users

       IV.  DESIGN CONSTRAINTS
v v v v   1.  Limited to 2 screens 24x80 characters
v v v v   2.  designed for an IBM PC
v v v v   3.  Imbedded in the dBase III environment
```

Figure 3.8 Example: Identification of the Needs (IDEAS)

process of this idea development includes the steps listed in Figure 3.7:

Figure 3.8 identifies more formally the needs of the project in J-Map notation.

3.4.2 REQUIREMENT SPECIFICATION

The requirement specification is the next section to be developed for the system. The contents of the requirements specifications include a complete software requirements specification. These features are listed in Figure 3.9.

1. Functions
2. Performance
3. External interfaces
4. Quality Attributes
5. Design Constraints

Figure 3.9 Requirement Specification Features

Good specifications require completeness, consistency, correctness, modifiability, measurability, traceability and process independence. Figure 3.10 is the requirement specification for part of this project.

```
REQUIREMENT SPECIFICATION:Function: Data entry/editor to support the J-Map notation

        I.  SYSTEM REQUIREMENTS
V . . .  1.  Develop a data entry procedure to support the J-Map notation.
. V . .  2.  Develop an editor to support the J-Map notation
. . V .  3.  Develop an interpreter
. . . V  4.  Develop a program generator
        II. FUNCTIONS
v v . .  1.  Data entry/editor to support the J-Map notation
. . v .  2.  Interpreter
. . . v  3.  Program generator
        III. PERFORMANCE
v v . .  1.  Reasonable speed for an interactive system
        IV. EXTERNAL INTERFACES
v v . .  1.  Storage of graphics screens in memory for "rapid"
             flipping between screens
        V.  QUALITY ATTRIBUTES
v v v .  1.  Incorporate structured software development techniques - dataflow,
             entity-relationships diagrams; structured charts, control flow graphics;
             structured development; walk-throughs.
. . . v  2.  Structured design of the program generated
        VI. DESIGN CONSTRAINTS
v v v v  1.  Limited to 2 screens 24x80 characters (1 graphics, 1 monochrome)
v v v v  2.  Designed for operation on an IBM PC
v v v v  3.  Imbedded in the dBase III environment
```

Figure 3.10 Example: Requirement Specification

3.4.3 DESIGN SPECIFICATION

A structured design provides the user with various forms of analysis at the design level and at the implementation level. The design specification allows the designer to review and walk-through the system. Techniques for analyzing software systems include control flow diagrams, external interrupts, state transition diagrams and entity relationships. The contents of the design specification are listed in Figure 3.11.

The similarity of contents between the requirement specification and the design specification differ in that the former is an overview perspective with the latter being very specific to the module. The example given in Figure

1. Functions
2. Performance
3. External interfaces
4. Internal interfaces
5. Quality Attributes
6. Design Constraints

Figure 3.11 Design Specification features

3.12 relates to that portion of the system dealing with the data entry/editor used to support the J-Map notation.

DESIGN SPECIFICATION:Function: Data entry/editor to support the J-Map notation	
	FUNCTIONS
V	1. Menu driven screens for selection of all activities
. V	2. Help available on request from all screens
. . V . . .	3. On screen editing available for J-Map notation
. . . V . .	4. Selection of specific items may be done with simple command constructs.
. . . . V .	5. Walk-throughs and reviews accomplished in the form of reviews of specific sections
. V	6. Verification and validation by computer analysis : testing for completeness
. V	7. Printed reports available for control flow, entity relationship diagrams, structured English, etc.
	PERFORMANCE
. . . . V .	1. Must be fast enough to not frustrate the user
	EXTERNAL INTERFACE
V V V V V .	1. Screen memory support software utilized to speed flipping through menu and help screens
	INTERNAL INTERFACE
V	1. Menus linked to main program
	QUALITY ATTRIBUTES
. V	1. User may select help option from anywhere within the system
V	2. Easy to learn and use
	DESIGN CONSTRAINTS
. . . . . V V	1. Database structure stores information - consequently this slows the system

Figure 3.12 Design Specification: Example: Data Entry/Editor to support J-Map notation

3.4.4 SOFTWARE SPECIFICATION

Software specifications become functionally more specific as refinement reaches the module level. Figure 3.13 outlines the features of the software specification. The example given in Figure 3.14 outlines the options available to the user in edit mode. It is this construct that provided the basis for the screen selection.

1. Identification of major features
2. Identification of elements
3. Relationships between major features and elements

Figure 3.13 Software Specification Features

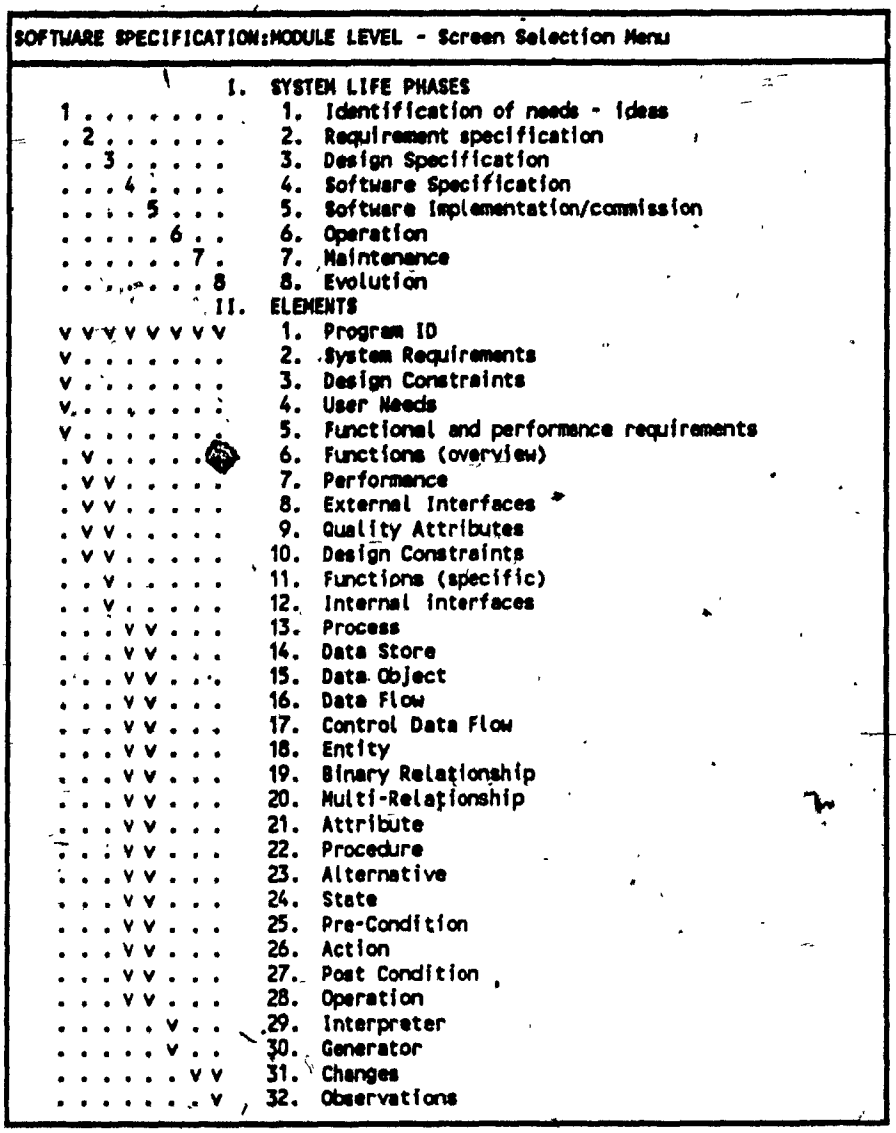


Figure 3.14 Software Specification:Example Module Level: Screen Selection Menu

3.4.5 SOFTWARE IMPLEMENTATION LEVEL

The design has been developed down to the software specification level. This provides considerable information

about the module(s) that is now going to be implemented.

Continuing with this example of the "Screen Selection Menu" the implementation will begin. The software specification identifies all selections available. The initial selection for this project is "SYSTEM LIFE PHASES"

1. Identification of cluster sets
2. Identification of elements of sets (alternatives)
3. Identification of conditions
4. Identification of operations
5. Identification of transactions on cluster sets
6. Identification of objects

Figure 3.15 Software Implementation Features

```
   SOFTWARE IMPLEMENTATION: Screen Selection Menu

v . . . . . 1.  Initialize screen, databases
. v . . . . 2.  Mark major areas to view
. . v . . . 3.  Selection of major areas complete
. . . v . . 4.  Mark sub-items to view
. . . . v . 5.  Selection of sub-items complete
. . . . . v 6.  Another major area requires marking of sub-items
. . . . . v 7.  All selecting complete
-------------------------------------------------------------------
s . . . . . 1.  Initialize screen, get databases
d s . . . . 2.  Select System Life Phase(s) to view
. d d s s d 3.  Display elements available for Phase;select elements
. . . d d s 4.  Are there more Phases?
. . . . . d 5.  Exit
-------------------------------------------------------------------
. f t . . . 1.  Phase_sel=.T.
. . . f t . 2.  Subsel_sel=.T.
. . . . . f t 3.  EOF()
-------------------------------------------------------------------
1 . . . . . 1.  Display Phases
2 2 . . . . 2.  read next selection
. 1 . . . . 3.  mark Phase record with "X"
. . 2 . . 1 4.  Display sub-items for current Phase
. . 3 2 . 2 5.  read next selection
. . . 1 . . 6.  mark sub-item record with "X"
. . 1 . 1 . 7.  get next marked Phase
```

Figure 3.16 Software Implementation :Example Screen 1; Selection Menu

(Figure 3.14). This becomes the first menu in the screen selection. Figure 3.15 describes the screen selection operation at the implementation level from a logical perspective. Refinement to the programming level would require the replacement of: 1) the action descriptions with a list of actions; 2) the guard descriptions with guard(s).

3.4.6 TESTING AND OPERATION PHASE

Inevitably there will be corrections and fine tuning of the system after implementation. Depending on the problems found, modification and additions may be required in the higher levels of the specification. Such things as speed and interfacing with the rest of the system are common areas requiring changes. Modifications should be noted indicating the specific function(s) (from the Design Specification Level) that may be affected. These changes should be logged in an organized manner. The contents of the testing and operation log should include the features listed in Figure 3.17.

1. Identification of the specific function (as defined in the design specification)
2. Description of change or modification
3. Performance
4. External interface
5. Quality Attributes
6. Design Constraints

Figure 3.17 Testing and Operations Log Features

Essentially this becomes an editing process of the design and software specification. The Software Implementation descriptions would also have to be modified. In this case, the original versions indicate additional changes have been made in the log. The log becomes a quick guide to changes made.

3.4.7 MAINTENANCE LEVEL

The maintenance level normally would entail the correction of errors found after implementation and minor revisions. These changes or corrections would be registered in a log such as the one described in the testing and operation phase. To simply modify without noting them could cause other errors to be generated in other sections of the program. These other errors may be difficult to correct without the knowledge of previous changes in other sections.

3.4.8 EVOLUTION LEVEL

Static systems probably don't exist or they are obsolete. Changes to a system can come in different forms. Major revisions would require a complete top down specification of the system. Portions of the existing specification may be valid but not necessarily. The evolution level often incorporates additions or enhancements to the existing system. Depending on the changes, additions and modifications to all levels of the specifications would

be required. This would be very similar to the development of the original specification. Again, a log noting what changes are taking place should be made in order to cope with any interface problems that may result.

3.5 CONCLUSIONS

Integrating information about all the levels of the software life cycle into an organized body of information becomes very much like a reference library. A card catalog becomes the key to a large vault of information. Likewise, the organized development of programs allows access to selected information much like that of a card catalog. Practice, refinement and organization are the requirements for the software designer.

The J-Map notation is based on sets, defined roles of sets and set members and associations between them. The notation supports control flow charts, data flow diagrams, structured pseudo descriptions as well as other popular structured software development techniques, including graphical representation.

4.0 DEVELOPING A PRODUCTIVE ENVIRONMENT

The most common representation of programs (Pascal, Basic, Fortran, etc) is a series of lines of code capable of repetition, selection and action. The structure of such programs are not particularly conducive to analysis and provides little help to the designer.

To support a designer with a productive environment requires integration of the many levels of program development. These levels would include:

1. Identification of needs
2. Requirement specification
3. Design specifications
4. Software specifications
5. Software implementation
6. Testing
7. Operation and Maintenance
8. Evolution

One of the more important requirements in this environment is that action at one level of program development is not destructive to the other levels. That is, program modification is integrated into the program specification and design in such a way that the specification and design documents are not out-of-sync even after modification.

The design of a specification must integrate levels of program development and management. Developing a specification with conceptual clarity and designed for maximum automation of systems analysis and automated code

generation requires an underlining logical process capable of supporting such properties.

4.1 A METHOD FOR DEFINING AND STORING PROGRAMS IN A DATABASE

The common programs (Pascal, Basic, Fortran, etc.) are represented in a format difficult for manipulations. Take for example the following program structure:

```
if (condition 1)
     if .not.(condition 3)
          action 2
     else
          if .not.(condition 2)
               action 1
          endif
     endif
else
     if .not.(condition 2)
          action 1
     endif
endif
```

Testing the program for completeness, consistency and redundancy in such a format is difficult.

Using a table format to design and represent systems provides a format that can be easily stored in a database. More importantly, many additional features can be incorporated into the support system. Computerized testing for completeness, consistency and redundancy can be performed. A measure of complexity can be calculated by computer analysis. Reports incorporating high level specifications and/or low level specifications can be

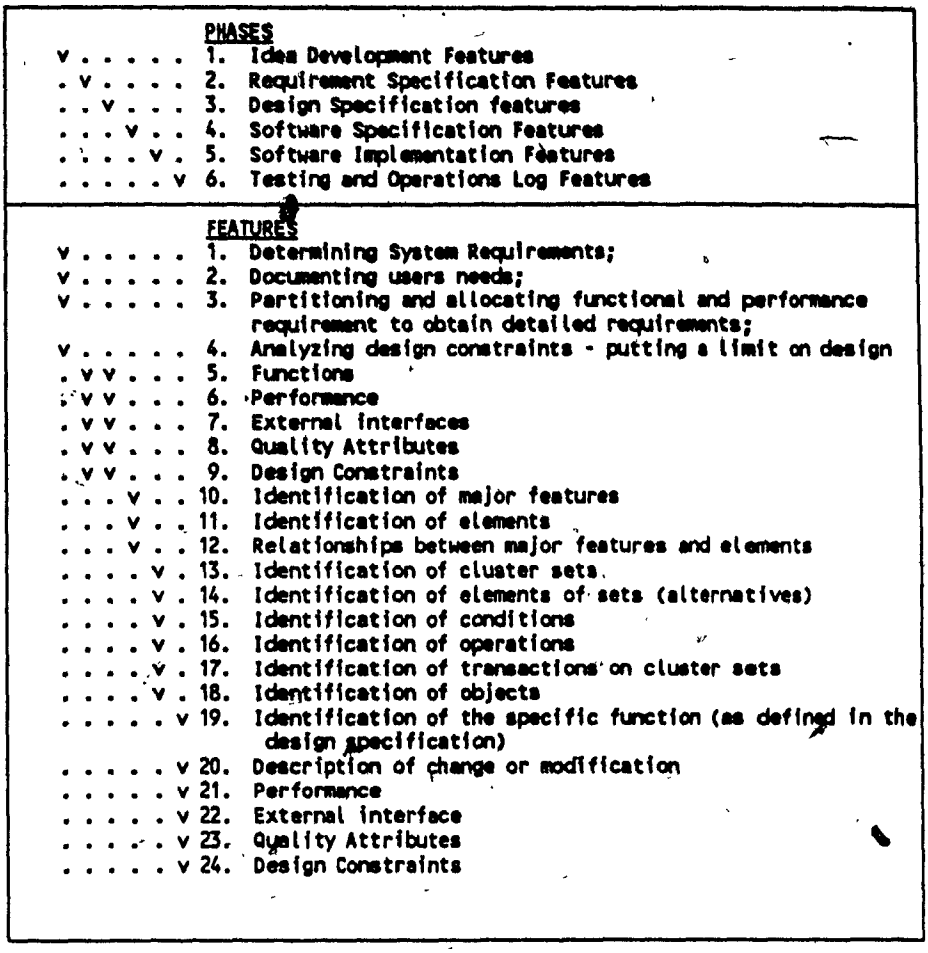


Figure 4.1 Levels of J-Map support in a System Life Cycle

produced. Analysis using database macros (join, select, etc.) can be performed.

4.2 PHASES AND FEATURES OF SYSTEM LIFE CYCLE

J-Maps view everything from a table map perspective. Table maps can be built from elements using the bottom up

method[21] or by functional decomposition as a top down approach. Figure 4.1 lists the phases of system development or the life cycle of software. The features of each phase are listed in the bottom part of the chart. Relationships between the phases and the features are shown in the columns.

4.3 SPECIAL OPTIONS AVAILABLE

The table notation allows for various types of tests and measures. Among the more important tests are those for completeness, consistency and redundancy. The table design allows for the testing of a complete set of alternatives at each decision point (cluster).

Many authors have expounded on the attributes of table representation. (A description of conventional decision tables is described in detail in Appendix A.) Decision tables are primitive J-Maps. The following are comments made by a few of them: "Structure tables provide a standard method for unambiguously describing complex, multi-variable, multi-result decision systems... Interestingly, however, one of the more promising application areas for structure tables appears to be in stating the logic of compilers and edit programs..."[22], "Any flowchart can be emulated by a decision table, whose complexity depends on that flowchart,"[23] "Decision tables can be used for the

description of parallel processes, which cannot be conveniently represented by flowcharts."[24]

The basic principles of decision tables are quite simple. One of the major advantages of the tables is that they provide a powerful means for expressing complex problems in greatly simplified form, which makes possible more effective analysis.

According to Dixon "Many basic advantages may be gained through the use of decision tables...including (1) cutting the analysis and programming time needed to develop and implement the program by as much as 50%, (2) greatly simplified updating procedures, and (3) drastic reductions in debugging effort, due to almost error-free development of the process logic"[25].

"The conversion of programs to decision tables is useful for debugging and optimization of programs (detection of static loops, dead-end rules and inaccessible instructions)"[26]. The detection of logical errors in decision table programs is defined by Ibramsha and Rajaraman[27]. Their method of detection is described in greater detail in the next section.

The development of specialized line and spreadsheet editors allow for the design and development of decision tables. These tools make decision table programming development practical.

4.3.1 COMPLETENESS, CONSISTENCY AND REDUNDANCY

Consider for instance overlapping conditions. In a fully expanded table, duplicate rules become obvious. In a composite rule table (a rule whose condition part consists of "t", "f", "-") this overlap may be obvious only to an experienced user:

v	v	v	(ALTERNATIVE) 1 2 3
l	s d	s d	(STATE) 1 2 3
- f -	t f -	t - f	(CONDITION) 1 2 3
x	x	x	(ACTION) 1 2

Figure 4.2 Overlapping Pairs of Alternatives

The overlapping pairs of alternatives are:

1 & 2 - redundant

2 & 3 - inconsistent (identical condition parts in overlapping expansions correspond to different actions A1 and A2

Intra-rule inconsistency occurs if the condition entry combinations are not valid. Consider the situation presented in Figure 4.3:

v	v	v	v	(ALTERNATIVE) 1 2 3 4
l	s d	s d	s d	(STATE) 1 2 3 4
t f f	f t f	t f t	f f t	(CONDITION) 1 Age <= 25 2 Age = 25 3 Age > 50
1	1	1 2		(ACTION) 1 2

Figure 4.3 Intra-rule Inconsistency

It is impossible for a person to have an age "x" satisfying the conditions in either column 2 (age=25 .and. not age <=25) or in column 3 (age<= 25 .and. not age = 25).

The detection of logical errors in decision table programs is defined by Ibramsha and Rajaraman.[28] They describe an algorithm to detect ambiguities and incomplete specifications in a decision table. The method is based on the use of a Karnaugh map. The program ANALYZER is based on this algorithm (in dBASE III+) and used for computer

assisted checking. The ANALYZER can be commanded by the user to find data sets which will lead to ambiguous or incomplete rules in a specified step. The ANALYZER will be described in greater detail in section 5.3.

4.3.2 MEASURE OF COMPLEXITY

The conversion of a J-Map Table to a tree-like structure is fairly elementary. From tree-like structures (or directly from the table) the characteristic Polynomial can be computed which may be used in complexity comparisons of algorithms. [29,30,31,32] (See section 5.2.1)

4.4 JMSS: J-MAP SUPPORT SYSTEM - AN OVERVIEW

The J-Map Support System is graphically illustrated in Figure 4.4. The most important feature to notice is the database storage of programs. This is the key to program analysis and report writing. Figure 4.5 models the features of the JMSS system to the types of reports that can be generated.

The editor edits the table structure described earlier. Several editors were designed and implemented for the editing process. However, Lotus 123 was found to be far more flexible during the implementation processes. The other development processes (identification of needs/requirement/design) and post development processes

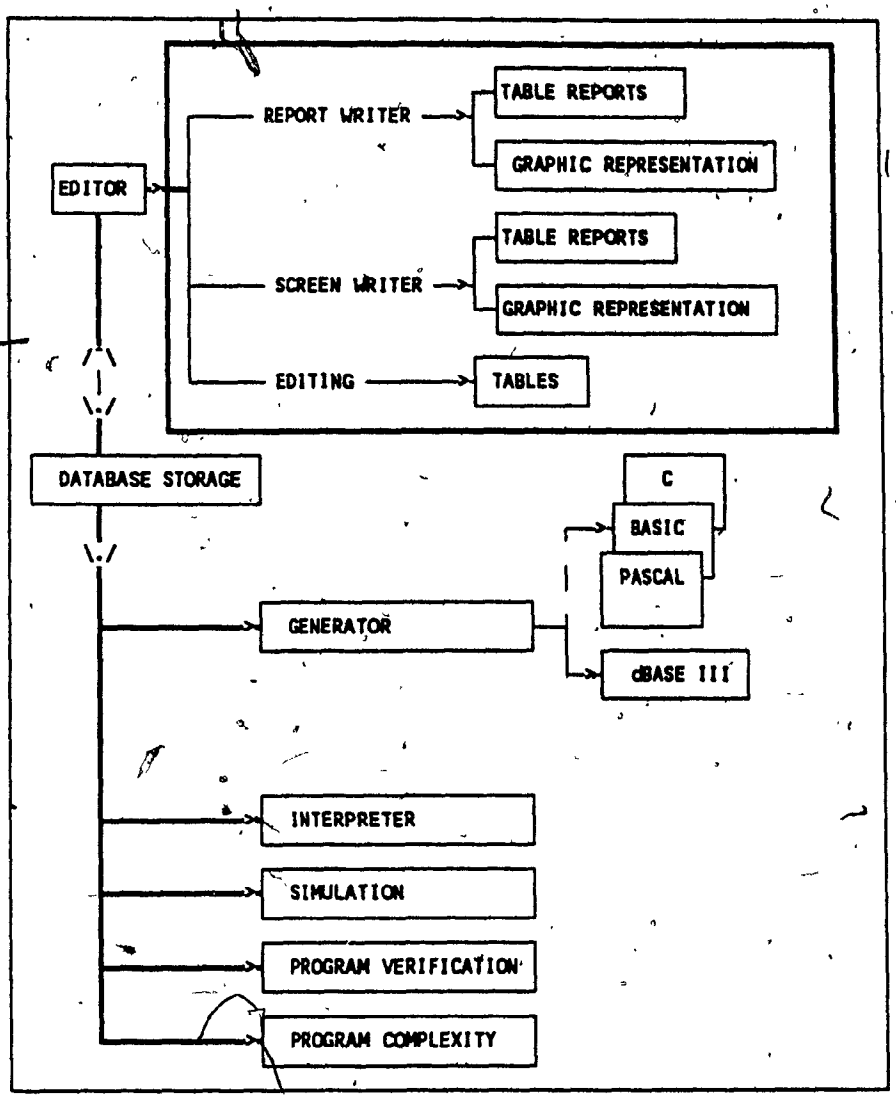


Figure 4.4 Graphical Model of J-Map Support System (JMSS)

were developed using the interactive dBase III application program. The implementation process is first developed in Lotus. The Lotus files are then transferred to dBase for report writing, program generation, etc. From the database information various tests may be made: program verification and program complexity.

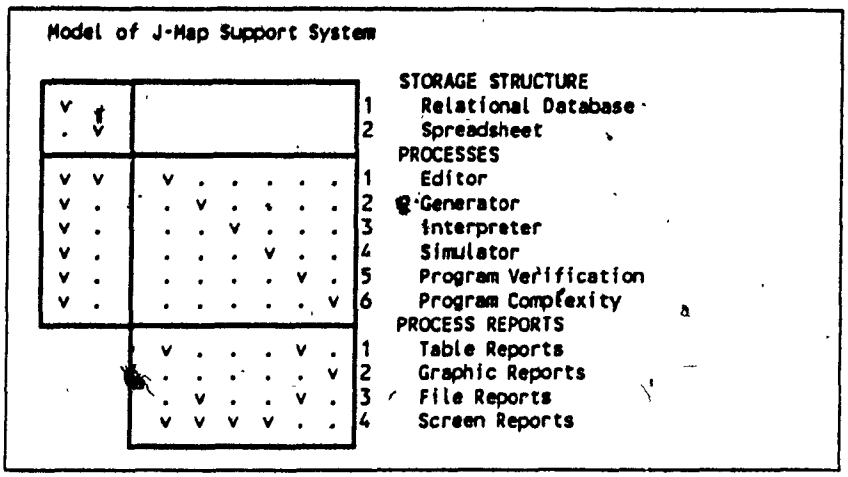


Figure 4.5 Model of J-Map Support System

The generator produces standard dBase III code from the database. The generator translates the information stored in the database into action lines of code. The structure applied is very simple as seen in Figure 4.6. The "case" structure operates quite efficiently in dBase resulting in reasonable execution speeds. Chapter 5 gives an example of a real time generated program. Other program generators could be developed to produce source code for other languages.

Other features of the JMSS system are the interpreter and the simulator. Both run using the table information stored in the database. The interpreter will run the

```
CLUSTER = 1
EXIT = .F.
DO WHILE .NOT. EXIT
     DO CASE
          CASE CLUSTER = 1
               *1.  Cluster 1 Description
               DO CASE
                    CASE  <Guard list of Alternative 1>
                              <Action list of Alternative 1>
                              CLUSTER = <next cluster>
                    CASE  <Guard list of Alternative 2>
                              <Action list of Alternative 2>
                              CLUSTER = <next cluster>
                              :
                              :
               ENDCASE
          CASE CLUSTER = 2
               *2.  Cluster 2 Description
               DO CASE
                    CASE  <Guard list of Alternative 1>
                              <Action list of Alternative 1>
                              CLUSTER = <next cluster>
                    CASE  <Guard list of Alternative 2>
                              <Action list of Alternative 2>
                              CLUSTER = <next cluster>
                              :
                              :
               ENDCASE
                    :
                    :
                    :
          ENDCASE
     ENDDO
```

Figure 4.6 Program Generator Output Structure

program in interpret mode to allow for the detection of errors. The simulation mode allows the designer to "walk-through" a

partially complete program.

The JMSS system became fairly large. It should be noted that this is a prototype system. Due to the limits of the Relation Database Manager System (dBASE III), (in particular speed), having a fully integrated system allowing interpretation while in the edit mode was possible but not practical with the old equipment available (4.7 MHz). It is

plausible that this combination would operate well on the modern IBM PS/2 Model 80 microcomputer that operates at a much faster speed.

The interpreter produces screen reports indicating the logical description of the current cluster and the currently selected alternative. As well, the current action is displayed. The program operates slowly enough that the operator can read these descriptions while the program was being interpreted. Getting from point A to point B in the program tends to be slow for large programs.

The simulator was developed in an attempt to overcome some of the shortcomings of the interpreter - i.e., looking at a particular cluster or set of clusters in the program and testing its operation. The weakness of the simulator became "what are the current values of variables?". This was not a particularly serious problem for simple programs or mathematical computation programs. However, when applied to database applications, the complexity expanded to include "what is the current database in use and what is the current record?". This became a serious problem when certain commands are used, in particular the "Locate ... Continue" commands. In dBASE, the locate command will locate a record in the database with specific attributes. The continue command will continue looking for other records having the same attributes in the database. Incorrect positioning of the database pointer, or carry over of an incorrect list of

locate attributes would give incorrect or unexpected results.

Table reports presenting selected parts of the program (see Chapter 5) are the preferred options for review of programs. These reports allow the review of selected information within the framework of the logical structure. Events leading up to the cluster as well as events that followed the cluster were often required in order to determine problems, errors or the next step. By searching for specific clusters, and/or attributes of the action, guard, alternative and/or cluster, informative reports could be produced from the database.

4.5 CONCLUSIONS

The environment developed for J-Map structures is a prototype for future systems. The integration of an editor, generator, interpreter and simulator are useful. Practical applications of the system for large implementations do have problems. Speed (on the older, slower IBM PC) was a handicap. However, this is not a problem on the newer PC's. The limitation of screen size was probably the most significant limitation. Program verification, complexity measures and graphical representations of programs aid in the design of logically correct systems. Additional features are explored and described in Chapter 5.

5.0 TOOLS FOR J-MAPS

5.1 INTRODUCTION

The fourth section described the use of J-Map constructs. Application of this methodology with pen and paper can and is being done. However, products are less than perfect with considerable cost in time. The pen and paper approach is not flexible enough as a design tool[33]. For example, an omission to a previous step results in a rewrite or a generally messy report. The tables tend to expand during development at all levels, particularly at the implementation level -- condition stub, action list, alternative selection as well as step level. To miscalculate the number of conditions at the beginning of the design results in several rewrites of the blueprint.

Computer assistance at the design level reduces the need to guess at the relative numbers of conditions, alternatives and steps required by the program. This frees the designer to concentrate on the task -- that is, the design of the program, not the implementation of the design. The implementation level of the design is by far the most dynamic in terms of size. Lotus 123 was used to develop the database because of its speed, flexibility and unique editing features. The printing features were also very useful. After initial implementation development in Lotus, data files were transferred to dBASE III Plus for refinement and analysis.

The main constraint of the computer assisted design tool is the system screen. The 24x80 characters provides the user with a very small view of his work. Overlays or windows provide the user with "flipping power" to be able to view necessary information quickly and easily. However, for large programming projects, printed reports are generally required. The limited screen size leaves a great deal to be desired. Higher resolution screens do help, but still lack the size and resolution of printed reports. It is for this reason, that greater emphasis is given to printed reports rather than screen displays. Numerous attempts were made to design a screen editor over the three year development of this thesis. These editors were useful for small problems but frustrating to use for larger problems. Perhaps with 24 inch, high resolution screens capable of 254 characters/line with 64 line screens running at a minimum of 20Mhz+, a truly useful editor can be developed. For now, however, the machines and screens available for this project were slow and small. Needless to say, working around the limitations became the real challenge.

5.2 THE DESIGNER TOOLS

"As the complexity and size of programs increase, the programmer is challenged with the task of organizing his program in a manner which will enhance intellectual

manageability. Thus, the structure and style are critical in regards to writing programs and verifying their correctness. In recent years, considerable emphasis has been placed on the correctness of programs and techniques for engineering them to be correct. However, more emphasis should be placed on designing languages which facilitate constructing correct programs."[34]

The development of quality programming habits must be supported by methods that appeal to the user's thinking processes at the highest level of the design phase. A methodology capable of overlaying high level programming languages with a set of features that stimulate problem-solving techniques will enhance the quality and correctness of programs.

The representation of programs in various ways also enhances the intellectual manageability of large and complex programs. Various methods of representation include graphical and table representation. The graphical representation provides a view of the complexity within a module. Table representation allows the verification of the correctness and completeness of the alternative at decision points. Another view of programs can be made by observing them in real time mode run with a special interpreters. Operating the module in interpretive mode allows review of the module during run time as well as a logical profile of the run. It is sometimes desirable to

run a module to examine its operation before the system is completely defined. This can be done by simulating manually the portion which is as-yet undefined or incomplete.

The tools developed for the support of the J-Map constructs are described in the following sections.

5.2.1 GRAPHICAL REPRESENTATION AND ANALYSIS OF IMPLEMENTATION LEVEL MODULES

The graphical representation of a module provides a view of the control flow. A complex graph represents a complex program. An example of a graph is given in Figure 5.1. Here alternatives are represented by rows, and clusters are represented by columns. From this graph, the complexity of the algorithm could be calculated based on Tamine[35] or the enhanced view by Wingerter[36]. Since the graphs themselves provide an indication of the complexity, the calculation of a number seems superfluous, though relatively easily done by weighting the number of decision points, the number of alternatives within each decision point, and the number of repetitions of decision points.

5.2.2 J-MAP TABLES REPRESENTING A PROGRAM

An example of a J-Map table representing a program is given in Figure 5.2. The program is large. Normally splitting the procedure into subroutines would be desirable. However, in dBase III Plus the "overhead" of subroutine calls was determined to be significant. Consequently, this

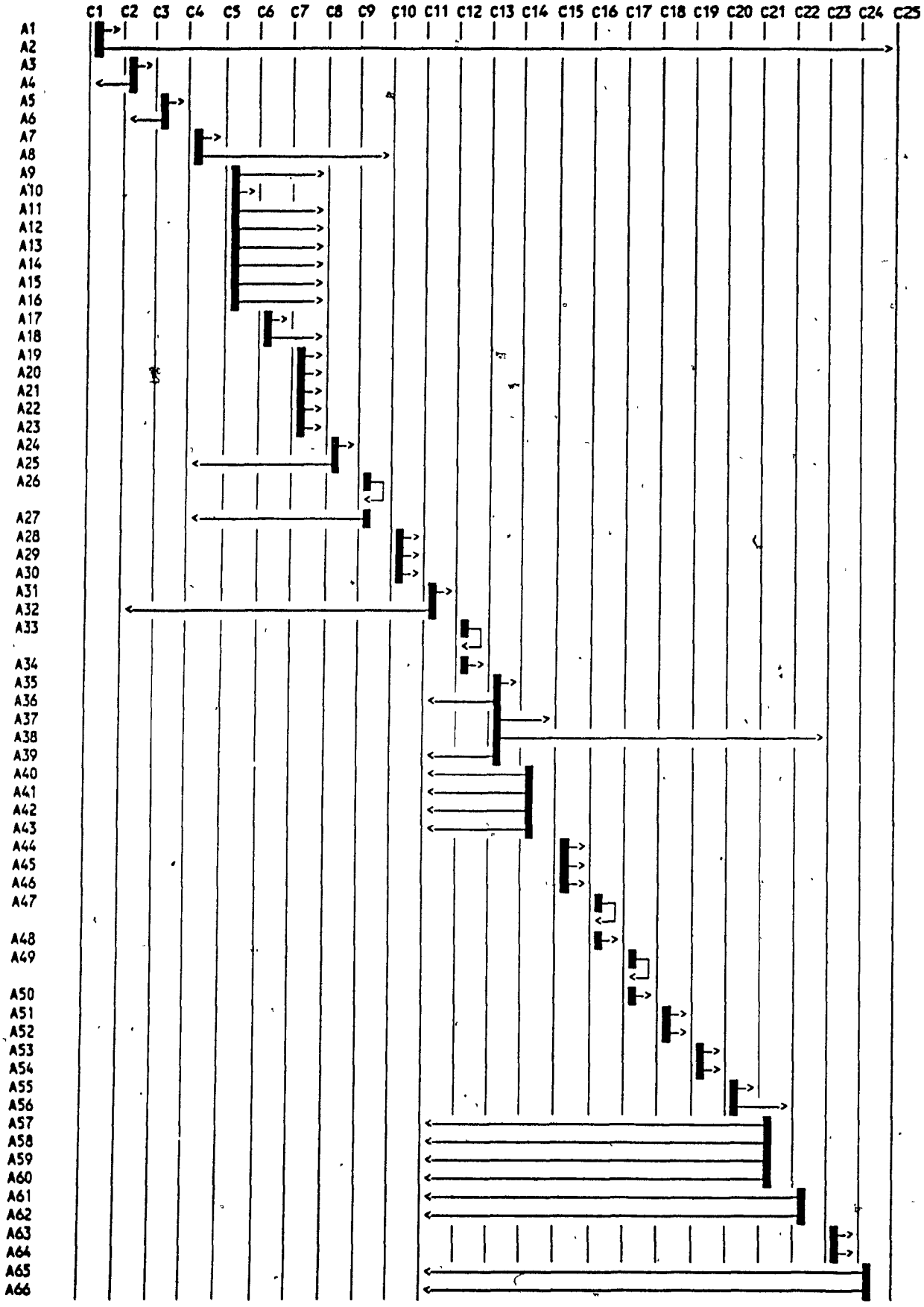
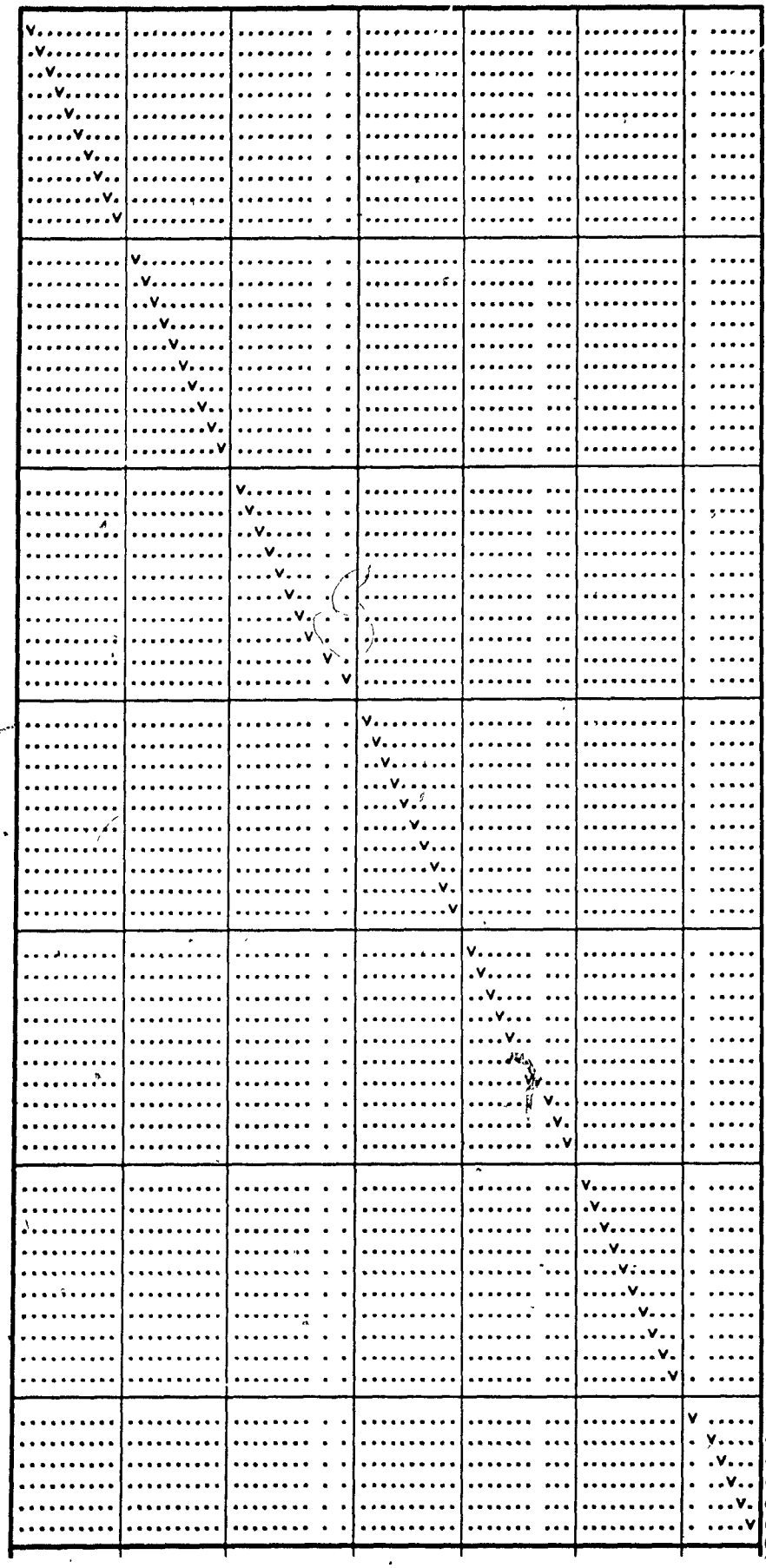
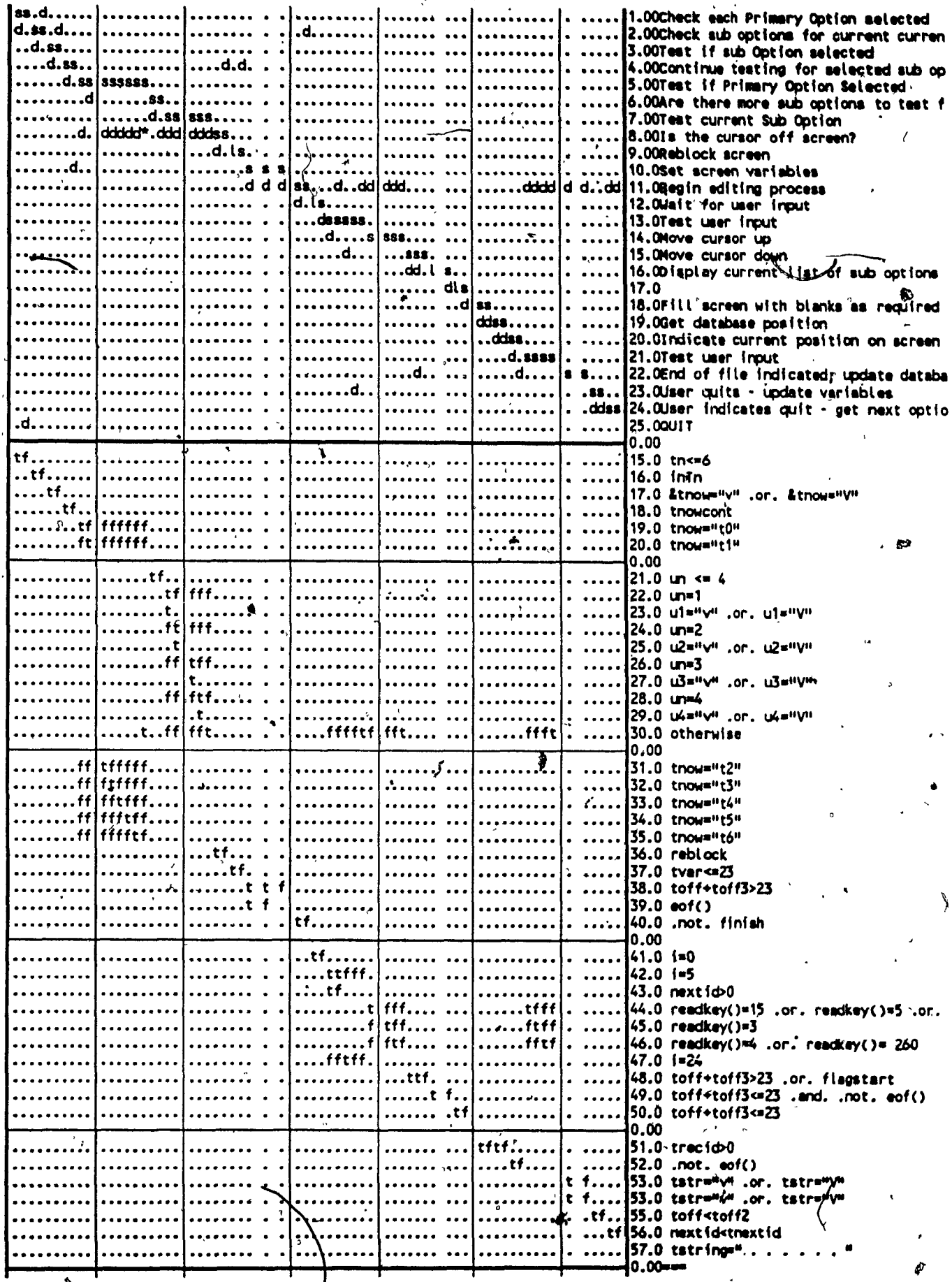


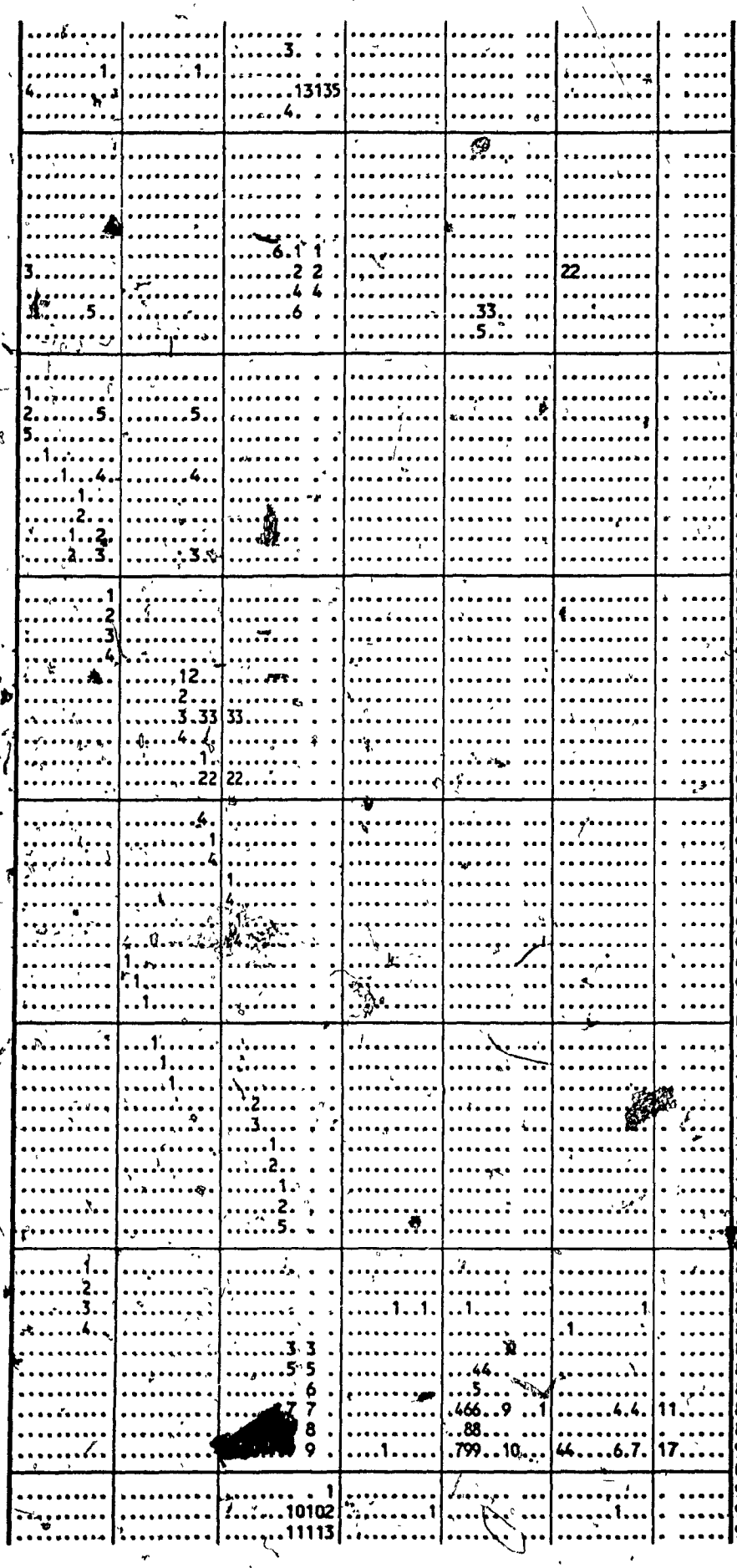
Figure 5.1 Computer Generated Graphical Representation of a Program "ABC"

real time example demonstrates a fairly large procedure (this is the same procedure displayed in graphic form in the previous section -Figure 5.1).

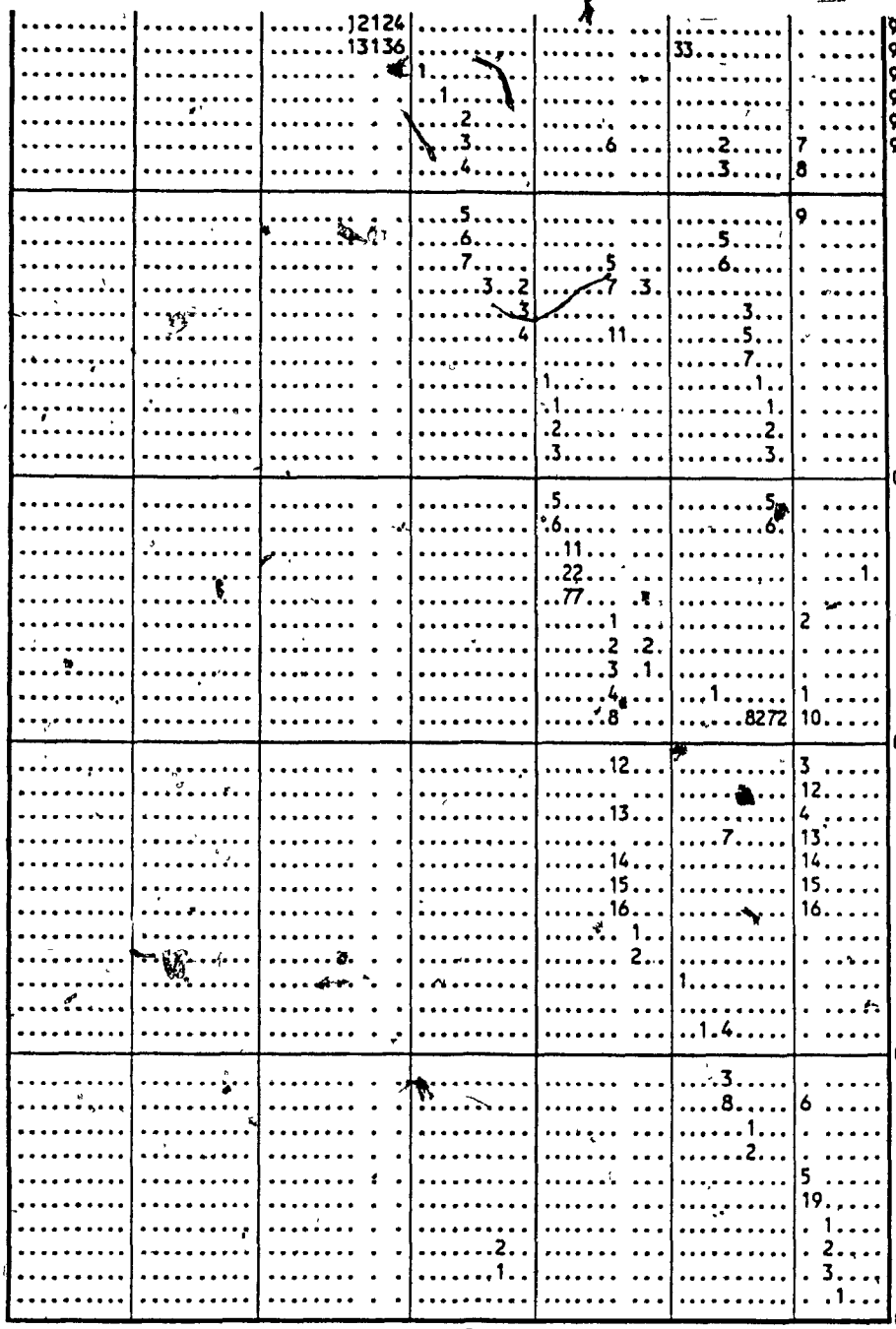
When procedures become as large and bulky, as this one has, information becomes more difficult to review. Subsets of the information becomes very helpful. Simple database commands can be used to select certain parts of the program. Figure 5.3 is an example viewing the first 8 alternatives composing the first four clusters of the example in 5.2. Figure 5.4 and 5.5 are other sets. Viewing only part of the program at at time is useful during development or debugging. It is not unlike conventional programming where one part of the program is debugged at a time.


0.00
1.0Current option in range
2.0All options have been tested:quit
3.0Current sub option to be tested
4.0All sub options for Primary option h
5.0Current sub option has been selected
6.0Current sub option has not been sele
7.0Test sub options:display current opt
8.0sub option editing complete; get nex
9.0t0: initialize flags
10.0t1:option selected
0.00
11.0t2:option selected
12.0t3:option selected
13.0t4:option selected
14.0t5:option selected
15.0t6:option selected
16.0None of the tn options selected
17.0
18.0
19.0
20.0
0.00
21.0
22.0
23.0
24.0Current Position off screen; reblock
25.0Reblocking screen not necessary
26.0Display current screen block
27.0Screen block complete-close block at
28.0End of list indicated
29.0List off screen- initialize variable
30.0Position still on screen
0.00
31.0Edit
32.0Editing complete for this item; get
33.0Wait for users action
34.0User action read
35.0User moved cursor up (not at top of
36.0User moved cursor up (currently at t
37.0User moved cursor down
38.0User finished
39.0User entry invalid - try again
40.0Next user action moves cursor down
0.00
41.0User Exits program
42.0Next user action moves cursor up
43.0Next user action invalid - try again
44.0
45.0
46.0
47.0Continue filling screen with data st
48.0Screen filled with items in memory
49.0Continue filling screen with blank f
50.0Screen complete
0.00
51.0
52.0
53.0Set db cursor position
54.0Current db position at beginning of
55.0Current item in file
56.0Currently at eof()
57.0Cursor moved down
58.0User exits
59.0Cursor moved up
60.0Next user action invalid - try again
0.00
61.0New record to be added to db file
62.0User ready to start next option
63.0Current column position within range
64.0Next column position out of range
65.0Next item id in range
66.0Next item id out of range
0.00


1.00Check each Primary Option selected
2.00Check sub options for current curren
3.00Test if sub Option selected
4.00Continue testing for selected sub op
5.00Test if Primary Option Selected
6.00Are there more sub options to test f
7.00Test current Sub Option
8.00Is the cursor off screen?
9.00Reblock screen
10.0Set screen variables
11.0Begin editing process
12.0Wait for user input
13.0Test user input
14.0Move cursor up
15.0Move cursor down
16.0Display current list of sub options
17.0
18.0Fill screen with blanks as required
19.0Get database position
20.0Indicate current position on screen
21.0Test user input
22.0End of file indicated; update databa
23.0User quits - update variables
24.0User indicates quit - get next optio
25.0QUIT
0.00
15.0 tn<=6
16.0 inkey
17.0 &tnow="v" .or. &tnow="V"
18.0 tnowcont
19.0 tnow="t0"
20.0 tnow="t1"
0.00
21.0 un <= 4
22.0 un=1
23.0 u1="v" .or. u1="V"
24.0 un=2
25.0 u2="v" .or. u2="V"
26.0 un=3
27.0 u3="v" .or. u3="V"
28.0 un=4
29.0 u4="v" .or. u4="V"
30.0 otherwise
0.00
31.0 tnow="t2"
32.0 tnow="t3"
33.0 tnow="t4"
34.0 tnow="t5"
35.0 tnow="t6"
36.0 reblock
37.0 tvar<=23
38.0 toff+toff3>23
39.0 eof()
40.0 .not. finish
0.00
41.0 i=0
42.0 i=5
43.0 nextid>0
44.0 readkey()=15 .or. readkey()=5 .or.
45.0 readkey()=3
46.0 readkey()=4 .or. readkey()= 260
47.0 i=24
48.0 toff+toff3>23 .or. flagstart
49.0 toff+toff3<=23 .and. .not. eof()
50.0 toff+toff3<=23
0.00
51.0 trecid>0
52.0 .not. eof()
53.0 tstr="v" .or. tstr="V"
53.0 tstr="v" .or. tstr="V"
55.0 toff<toff2
56.0 nextid<nextid
57.0 tstring=". . . . . "
0.00==



```
21.0 toff=6
22.0 toff2=1
23.0 tcol=4
24.0 trow=2
25.0 nextid=0
0.00
31.0 zn=0
32.0 tnow="t0"
33.0 t0="v"
34.0 tn=0
35.0 arow=trow+2
36.0 reblock=.f.
37.0 toff=1
38.0 tstart=1
39.0 tid=nextid
40.0 trecid=0
0.00
41.0 trec10=1
42.0 @ 23,45 SAY "TSTOP = 1 "
43.0 reblock=.t.
44.0 inTm=.t.
45.0 @ 23,45 SAY "TSTOP = 2 "
46.0 tnowcont=.t.
47.0 @ 23,45 SAY "TSTOP = 3 "
48.0 @ trow,2*tspace+7
49.0 tn=tn+1
50.0 tnow="t"+ltrim(str(tn))
0.00
51.0 @ trow,tcol say "v"
52.0 @ arow,tcol say "v"
53.0 @ 4+tot1,2 say chr(195)+substr(sstr
54.0 @ trow,2*tspace+7 say "1. IDEA Deve
55.0 un=un+1
56.0 toff3=tot1 + 4
57.0 tnowcont=.f.
58.0 unow="u"+ltrim(str(un))
59.0 @ arow,2*tspace+7 say "1. Determine
60.0 arow=arow+1
0.00
61.0 set filter to field_type="U1"
62.0 @ arow,2*tspace+7 say "2. Document
63.0 set filter to field_type="U2"
64.0 @ arow,2*tspace+7 say "3. Partition
65.0 set filter to field_type="U3"
66.0 @ arow,2*tspace+7 say "4. Determine
67.0 set filter to field_type="U4"
68.0 @ 22,40 say "TNOW='T2'"
69.0 @ 22,40 SAY "TNOW='T3'"
70.0 @ 22,40 SAY "TNOW='T4'"
0.00
71.0 @ 22,40 SAY "TNOW='T5'"
72.0 @ 22,40 SAY "TNOW='T6'"
73.0 @ 22,40 say "error in 't' value"
74.0 tvar=1
75.0 @ tvar,2 say chr(218)+substr(sstrin
76.0 @ tvar+1,2 say chr(179)+substr(tstr
77.0 tvar=tvar+1
78.0 @ 3,2 say chr(195)+substr(sstring,1
79.0 @ 24,2 say chr(192)+substr(sstring,
80.0 tnextid=0
0.00
81.0 tcontinue=.t.
82.0 finish=.f.
83.0 i=0
84.0 goto top
85.0 nextid=tnextid+1
86.0 @ 2,45 say "rec_no="+ltrim(str(rec_
87.0 trecid=rec_no
88.0 sele 10
89.0 trec10=rec_no
90.0 sele 1
0.00
91.0 tstart=toff
92.0 i=24
93.0 flagstart=.t.
```



```
94.0 toff=tstart
95.0 nextid=tid
96.0 @ 23,45 SAY "TSTOP = 5 "
97.0 i=inkey()
98.0 seek trec
99.0 @ toff+toff3,2*tspace+7 say ltrim(s
 100 @ toff+toff3,2*tspace+10 get desc
 101 read
 102 @ 23,40 say "readkey()="+str(readke
 103 @ toff+toff3,2*tspace+10 say desc
 104 toff=toff+1
 105 nextid=nextid + 1
 106 skip
 106 skip
 107 i=6
 108 i=5
 109 toff=toff-1
 110 nextid=nextid - 1
0.00
 111 skip -1
 112 trec=file_rec_n
 113 flagstart=.f.
 114 nextid=tnextid
 115 goto bottom
 116 nextid=nextid+1
 117 @ toff+toff3,2 say chr(179)+substr(
 118 @ toff+toff3,2+tspace+7
 119 @ toff+toff3,tcol say "v"
 120 trec=recno()
0.00
 121 APPEND BLANK
 121 append blank
 122 REPLACE REC_NO WITH RECNO()
 122 replace rec_no with recno()
 123 REPLACE REC_TYPE WITH "T1"
 124 REPLACE FILE_SEL WITH "SELE 1"
 125 REPLACE FILE_REC_N WITH TREC
 126 toff2=toff
 127 tnextid=nextid
 128 seek trec10
 129 nextid=tid + 1
 130 seek trecid
0.00
 131 READ
 132 replace id with nextid
 133 tstr="."
 134 @ toff+toff3,tcol get tstr
 135 replace field_type with unow
 136 @ toff+toff3,tcol say tstr
 137 @ toff+toff3,tcol say "."
 138 tcol=tcol+2
 139 finish=.t.
 140 toff=toff2
0.00
```

Figure 5.2 Computer Generated Table Representation Program "ABC"

List all for clusters C1,C2,C3 and C4 with table, id, and desc

```
          0.00
v.......  1.00Current option in range
.v......  2.00All options have been tested:quit
..v.....  3.00Current sub option to be tested
...v....  4.00All sub options for Primary option have been tested:get next Option
....v...  5.00Current sub option has been selected
.....v..  6.00Current sub option has not been selected; get next sub option
......v.  7.00Test sub options:display current option and sub option
.......v  8.00sub option editing complete; get next option
          0.00
ss.d....  1.00Check each Primary Option selected
d.ss.d..  2.00Check sub options for current current Primary
..d.ss..  3.00Test if sub Option selected
....d.ss  4.00Continue testing for selected sub options
......d.  5.00Test if Primary Option Selected
.......d  10.0Set screen variables
.d......  25.0Quit
          0.00
tf......  15.0 tn<=6
..tf....  16.0 inTn
....tf..  17.0 &tnow="v" .or. &tnow="V"
......tf  18.0 tnowcont
          0.00
4.......  24.0 trow=2
3.......  37.0 toff=1
......5.  39.0 tid=nextid
1.......  42.0 @ 23,45 SAY "TSTOP = 1 "
2.......  43.0 reblock=.t.
5.......  44.0 inTn=.t.
..1.....  45.0 @ 23,45 SAY "TSTOP = 2 "
....1...  46.0 tnowcont=.t.
.....1..  47.0 @ 23,45 SAY "TSTOP = 3 "
......2.  48.0 @ trow,2*tspace+7
.....1..  49.0 tn=tn+1
.....2..  50.0 tnow="t"+ltrim(str(tn))
.......1  81.0 tcontinue=.t.
.......2  82.0 finish=.f.
.......3  83.0 i=0
.......4  84.0 goto top
          0.00
```

Figure 5.3 Selection of the First Four Clusters From Figure 5.2

List all for cluster C5 with table, fd, and desc

```
        0.00
v.......9.00t0: initialize flags
.v......10.0t1:option selected
..v.....11.0t2:option selected
...v....12.0t3:option selected
....v...13.0t4:option selected
.....v..14.0t5:option selected
......v.15.0t6:option selected
.......v16.0None of the tn options selected
        0.00
ssssssss5.00Test if Primary Option Selected
.d......6.00Are there more sub options to test for?
d.ddddd*8.00Is the cursor off screen?
        0.00
tfffffff19.0 tnow="t0"
ftffffff20.0 tnow="t1"
.......t30.0 otherwise
fftfffff31.0 tnow="t2"
fffftfff32.0 tnow="t3"
ffffffff33.0 tnow="t4"
fffffftf34.0 tnow="t5"
ffffffft35.0 tnow="t6"
        0.00
1.......23.0 tcol=4
5.......43.0 reblock=.t.
4.......46.0 tnowcont=.t.
2.......49.0 tn=tn+1
3.......50.0 tnow="t"+ltrim(str(tn))
.1......51.0 @ trow,tcol say "v"
.2......52.0 @ arow,tcol say "v"
.3......53.0 @ 4+tot1,2 say chr(195)+substr(sstring,1,2*tspace+1)+chr(180)
.4......54.0 @ trow,2*tspace+7 say "1. IDEA Development Features"
..1.....68.0 @ 22,40 say "TNOW='T2'"
...1....69.0 @ 22,40 SAY "TNOW='T3'"
....1...70.0 @ 22,40 SAY "TNOW='T4'"
.....1..71.0 @ 22,40 SAY "TNOW='T5'"
......1.72.0 @ 22,40 SAY "TNOW='T6'"
.......173.0 @ 22,40 say "error in 't' value"
        0.00
```

Figure 5.4 Selection of the Cluster Five From Figure 5.2

List all for cluster C7 with entrance and exit clusters with table, id, and desc

```
          0.00
v.......  17.0
.v......  18.0
..v.....  19.0
...v....  20.0
....v...  21.0
.....v..  22.0
......v.  23.0
.......v  24.0Current Position off screen; reblock screen
          0.00
ss......  6.00Are there more sub options to test for?
d.ssssss  7.00Test current Sub Option
.ddddddd  8.00Is the cursor off screen?
.......d  9.00Reblock screen
          0.00
tf......  21.0 un <= 4
..fffff.  22.0 un=1
..t.....  23.0 u1="V" .or. u1="V"
..ftfff.  24.0 un=2
...t....  25.0 u2="V" .or. u2="V"
..fftff.  26.0 un=3
....t...  27.0 u3="V" .or. u3="V"
..ffftf.  28.0 un=4
.....t..  29.0 u4="V" .or. u4="V"
..fffft.  30.0 otherwise
.......t  36.0 reblock
          0.00==
.1......  23.0 tcol=4
.5......  43.0 reblock=.t.
.4......  46.0 tnowcont=.t.
.3......  50.0 tnow="t"+ltrim(str(tn))
12......  55.0 un=un+1
2.......  56.0 toff3=tot1 + 4
3.3333..  57.0 tnowcont=.f.
4.......  58.0 unow="u"+ltrim(str(un))
..1.....  59.0 @ arow,2*tspace+7 say "1. Determine System Requirements"
..2222..  60.0 arow=arow+1
..4.....  61.0 set filter to field_type="U1"
...1....  62.0 @ arow,2*tspace+7 say "2. Document Users Needs"
...4....  63.0 set filter to field_type="U2"
....1...  64.0 @ arow,2*tspace+7 say "3. Partition/Allocate functional requirements"
....4...  65.0 set filter to field_type="U3"
.....1..  66.0 @ arow,2*tspace+7 say "4. Determine System Requirements"
.....4..  67.0 set filter to field_type="U4"
.......2  74.0 tvar=1
          0.00
```

Figure 5.5 Selection of the Clusters Six, Seven and Part of Eight From Figure 5.2

5.2.3 CONTROL STRUCTURES

The condition stub for each alternative within a cluster controls the selection of the alternative. Only one alternative will meet the match of the current state. Control flow is directed to the alternative that matches the current state. Actions of the alternative are sequentially executed in the order specified. At the end of the series of actions the cluster block indicates the next cluster to be selected (with a value of "d" or "l" for repetition).

5.2.4 DATA FLOW REVIEW

Database selection provides a useful tool in reviewing selected variables used in a program. The table structure allows review of modification of variables, and notes where this modification occurs. Various types of errors can be detected: duplicate actions, modification of variables without being referenced, lack of initialization, etc.

The database selection allows for the trimming of unnecessary information. During the development selected types of information may be of interest. Figures

List all parts related to variable "tid", "nextid" and "tnextid"
in cluster 1-9

			0.00
v.....	.....	.....	1.00Current option in range
.v.....	.....	.....	2.00All options have been tested:quit
..v.....	.....	.....	3.00Current sub option to be tested
...v.....	.....	.....	4.00All sub options for Primary option have been tested:get next Option
....v.....	.....	.....	5.00Current sub option has been selected
.....v....	.....	.....	6.00Current sub option has not been selected; get next sub option
......v...	.....	.....	7.00Test sub options:display current option and sub option
.......v..	.....	.....	8.00sub option editing complete; get next option
........v.	.....	.....	9.00t0: initialize flags
.........v	.....	.....	10.0t1:option selected
..........v	.....	.....	11.0t2:option selected
..........v	.....	.....	12.0t3:option selected
..........v	.....	.....	13.0t4:option selected
..........v	.....	.....	14.0t5:option selected
..........v	.....	.....	15.0t6:option selected
..........v	.....	.....	16.0None of the tn options selected
..........v	.....	.....	17.0
..........v	.....	.....	18.0
..........v	.....	.....	19.0
..........v	.....	.....	20.0
..........v	.....	.....	21.0
..........v	.....	.....	22.0
..........v	.....	.....	23.0
..........v	.....	.....	24.0Current Position off screen; reblock screen
..........v	.....	.....	25.0Reblocking screen not necessary
..........v	.....	.....	26.0Display current screen block
..........v	.....	.....	27.0Screen block complete-close block at ends
			0.00
ss.d.....	.....	.....	1.00Check each Primary Option selected
d.ss.d....	.....	.....	2.00Check sub options for current current Primary
..d.ss....	.....	.....	3.00Test if sub Option selected
...d.ss..	.....	....d.d	4.00Continue testing for selected sub options
.....d.ss	ssssss....	.....	5.00Test if Primary Option Selected
.........d	.....ss..	.....	6.00Are there more sub options to test for?
....	.....d.ss	sss....	7.00Test current Sub Option
.....d.	ddddd*.ddd	ddd.s..	8.00Is the cursor off screen?
....	.....	.....ls	9.00Reblock screen
.....d..	.....	.....	10.0Set screen variables
.....	.....	.....	11.0Begin editing process
.....	.....	...s...	12.0Wait for user input
.....	.....	...d...	13.0Test user input
.d.....	.....	.....	25.0EDIT
			0.00
.....	.....	.....	43.0 nextid>0
.....	.....	.....	56.0 nextid<tnextid
			0.00
.....	.....	4	25.0 nextid=0
.....5..	.....	.....	39.0 tid=nextid
.....	.....	5	80.0 tnextid=0
.....	.....	.....	85.0 nextid=tnextid+1
.....	.....	.....	95.0 nextid=tid
.....	.....	.....	99.0 @ toff+toff3,2*tspace+7 say ltrim(str(nextid))+"."
.....	.....	.....	105 nextid=nextid + 1
.....	.....	.....	110 nextid=nextid - 1
.....	.....	.....	114 nextid=tnextid
.....	.....	.....	116 nextid=nextid+1
.....	.....	.....	127 tnextid=nextid
.....	.....	.....	129 nextid=tid + 1
.....	.....	.....	132 replace id with nextid

Figure 5.6 Selected Dataflow Elements in Clusters 1-9: tid, nextid, tnextid

List all parts related to variable "tid", "nextid" and "tnextid"
in cluster 10-17

```
v . .|..........|..........| 28.0End of list indicated
. v .|..........|..........| 29.0List off screen- initialize variables
. . v|..........|..........| 30.0Position still on screen
. . .|v.........|..........| 31.0Edit
. . .|.v........|..........| 32.0Editing complete for this item; get next item
. . .|..v.......|..........| 33.0Wait for users action
. . .|...v......|..........| 34.0User action read
. . .|....v.....|..........| 35.0User moved cursor up (not at top of page)
. . .|.....v....|..........| 36.0User moved cursor up (currently at top of page)
. . .|......v...|..........| 37.0User moved cursor down
. . .|.......v..|..........| 38.0User finished
. . .|........v.|..........| 39.0User entry invalid - try again
. . .|.........v|..........| 40.0Next user action moves cursor down
. . .|..........|v.........| 41.0User Exits program
. . .|..........|.v........| 42.0Next user action moves cursor up
. . .|..........|..v.......| 43.0Next user action invalid - try again
. . .|..........|...v......| 44.0
. . .|..........|....v.....| 45.0
. . .|..........|.....v....| 46.0
. . .|..........|......v...| 47.0Continue filling screen with data stored
. . .|..........|.......v..| 48.0Screen filled with items in memory
. . .|..........|........v.| 49.0Continue filling screen with blank fill
. . .|..........|.........v| 50.0Screen complete
-----+----------+----------+-------------------------------------------------
. . .|.d........|..........| 2.00Check sub options for current current Primary
s s s|..........|..........| 10.0Set screen variables
d d d|ss...d..dd|ddd.......| 11.0Begin editing process
. . .|d.ls......|..........| 12.0Wait for user input
. . .|...dssssss|...-......| 13.0Test user input
. . .|....d....s|sss.......| 14.0Move cursor up
. . .|......d...|...sss....| 15.0Move cursor down
. . .|..........|...dd.ls..| 16.0Display current list of sub options to be edited
. . .|..........|.......dls| 17.0
. . .|..........|.........d| 18.0Fill screen with blanks as required
. . .|..........|....d.....| 22.0End of file indicated; update database with new entry
. . .|.......d..|..........| 23.0User quits - update variables
-----+----------+----------+-------------------------------------------------
                                0.00
-----+----------+----------+-------------------------------------------------
. . .|....tf....|..........| 43.0 nextid>0
. . .|..........|..........| 56.0 nextid<tnextid
-----+----------+----------+-------------------------------------------------
. . .|..........|..........| 25.0 nextid=0
6 . .|..........|...33.....| 39.0 tid=nextid
. . .|..........|..........| 80.0 tnextid=0
3 3 .|..........|..........| 85.0 nextid=tnextid+1
13136|..........|..........| 95.0 nextid=tid
. ...|....3.....|......6...| 99.0 @ toff+toff3,2*tspace+7 say ltrim(str(nextid))+"."
. . .|.........3|..........|  105 nextid=nextid + 1
. . .|..........|.3........|  110 nextid=nextid - 1
. . .|..........|..22......|  114 nextid=tnextid
. . .|..........|......1...|  116 nextid=nextid+1
. . .|..........|.......2..|  127 tnextid=nextid
. . .|..........|..........|  129 nextid=tid + 1
. . .|..........|..........|  132 replace id with nextid
```

Figure 5.7 Selected Dataflow Elements in Clusters 10-17: tid, nextid, tnextid

List all parts related to variable "tid", "nextid" and "tnextid"
in cluster 18-24

```
                                                0.00
v.........|. .....|51.0
.v........|. .....|52.0
..v.......|. .....|53.0Set db cursor position
...v......|. .....|54.0Current db position at beginning of file
....v.....|. .....|55.0Current item in file
.....v....|. .....|56.0Currently at eof()
......v...|. .....|57.0Cursor moved down
.......v..|. .....|58.0User exits
........v.|. .....|59.0Cursor moved up
.........v|. .....|60.0Next user action invalid - try again
----------|- -----|0.00
..........|v .....|61.0New record to be added to db file
..........|. v....|62.0User ready to start next option
..........|. .v...|63.0Current column position within range- move to next column
..........|. ..v..|64.0Next column position out of range
..........|. ...v.|65.0Next item id in range
..........|. ....v|66.0Next item id out of range
                                                0.00
......dddd|d d..dd|11.0Begin editing process
ss........|. .....|18.0Fill screen with blanks as required
ddss......|. .....|19.0Get database position
..ddss....|. .....|20.0Indicate current position on screen
....d.ssss|. .....|21.0Test user input
.....d....|s s....|22.0End of file indicated; update database with new entry
..........|. .ss..|23.0User quits - update variables
..........|. .ddss|24.0User indicates quit - get next option
                                                0.00
..........|. .....|43.0 nextid>0
..........|. ...tf|56.0 nextid<tnextid
                                                0.00
..........|. .....|25.0 nextid=0
..........|. .....|39.0 tid=nextid
..........|. .....|80.0 tnextid=0
..........|. .....|85.0 nextid=tnextid+1
33........|. .....|95.0 nextid=tid
....2.....|7 .....|99.0 @ toff+toff3,2*tspace+7 say ltrim(str(nextid))+"."
......3...|. .....| 105 nextid=nextid + 1
........3.|. .....| 110 nextid=nextid - 1
..........|. ...1.| 114 nextid=tnextid
..........|2 .....| 116 nextid=nextid+1
..........|. .....| 127 tnextid=nextid
..........|. .....| 129 nextid=tid + 1
....8.....|6 .....| 132 replace id with nextid
                                                0.00
```

Figure 5.8 Selected Dataflow Elements in clusters 18-24: tid, nextid, tnextid

5.6, 5.7 and 5.8 display those areas where variables "tid", "nextid", and "tnextid" are referenced or modified. Each display is for a different set of clusters. Figures 5.9, 5.10 and 5.11 show where the databases are modified in the program.

These reports are relatively easy to generate by simple database manipulation. They are unique relative to reports found in conventional programming. Knowing where a problem is with respect to the rest of the program in a logical sense reduces the amount of ripple errors that occur when a correction is made in one location with effects felt in others. This type of report is a practical approach to practical problems. Other types of reports could be produced as the need arises.

Some methodologies approach dataflow at a much more basic level. Dataflow becomes more an assembly language exercise rather than a high level approach to programming. In keeping with higher level languages, the level of detail is reduced to a practical level, not a machine level.

List all parts related to database commands: Replace, Append, Select in the first 7 cluster.

```
                                                    0.00
v.........  ..........  ...  1.00Current option in range
.v........  ..........  ...  2.00All options have been tested:quit
..v.......  ..........  ...  3.00Current sub option to be tested
...v......  ..........  ...  4.00All sub options for Primary option have been tested:get next Option
....v.....  ..........  ...  5.00Current sub option has been selected
.....v....  ..........  ...  6.00Current sub option has not been selected; get next sub option
......v...  ..........  ...  7.00Test sub options:display current option and sub option
.......v..  ..........  ...  8.00sub option editing complete; get next option
........v.  ..........  ...  9.00t0: initialize flags
.........v  ..........  ...  10.0t1:option selected
                                                    0.00
..........  v.........  ...  11.0t2:option selected
..........  .v........  ...  12.0t3:option selected
..........  ..v.......  ...  13.0t4:option selected
..........  ...v......  ...  14.0t5:option selected
..........  ....v.....  ...  15.0t6:option selected
..........  .....v....  ...  16.0None of the tn options selected
..........  ......v...  ...  17.0
..........  .......v..  ...  18.0
..........  ........v.  ...  19.0
..........  .........v  ...  20.0
                                                    0.00
..........  ..........  v..  21.0
..........  ..........  .v.  22.0
..........  ..........  ..v  23.0
                                                    0.00
ss.d......  ..........  ...  1.00Check each Primary Option selected
d.ss.d....  ..........  ...  2.00Check sub options for current current Primary
..d.ss....  ..........  ...  3.00Test if sub Option selected
....d.ss..  ..........  ...  4.00Continue testing for selected sub options
......d.ss  ssssss....  ...  5.00Test if Primary Option Selected
.........d  ......ss..  ...  6.00Are there more sub options to test for?
..........  ......d.ss  sss  7.00Test current Sub Option
........d.  ddddd*.ddd  ddd  8.00Is the cursor off screen?
..........  ..........  ...  9.00Reblock screen
.......d..  ..........  ...  10.0Set screen variables (
.d........  ..........  ...  25.0EXIT
                                                    0.00
..........  ..........  ...  88.0 sele 10
..........  ..........  ...  90.0 sele 1
..........  ..........  ...   121 APPEND BLANK
..........  ..........  ...   121 append blank
..........  ..........  ...   122 REPLACE REC_NO WITH RECNO()
..........  ..........  ...   122 replace rec_no with recno()
..........  ..........  ...   123 REPLACE REC_TYPE WITH "T1"
..........  ..........  ...   124 REPLACE FILE_SEL WITH "SELE 1"
..........  ..........  ...   125 REPLACE FILE_REC_N WITH TREC
..........  ..........  ...   132 replace id with nextid
..........  ..........  ...   135 replace field_type with unow
                                                    0.00
```

Figure 5.9 Selected Dataflow Elements: Modification to the Database in the First 7 Clusters (note:no changes)

List all parts related to database commands: Replace, Append, Select in cluster 8-14.

```
v.... . . .......... ... 24.0Current Position off screen; reblock screen
.v...  . . .......... ... 25.0Reblocking screen not necessary
..v.. . . .......... ... 26.0Display current screen block
...v. . . .......... ... 27.0Screen block complete-close block at ends
....v . . .......... ... 28.0End of list indicated
..... v . .......... ... 29.0List off screen- initialize variables
..... . v .......... ... 30.0Position still on screen
                         0.00
..... . . v......... ... 31.0Edit
..... . . .v........ ... 32.0Editing complete for this item; get next item
..... . . ..v....... ... 33.0Wait for users action
..... . . ...v...... ... 34.0User action read
..... . . ....v..... ... 35.0User moved cursor up (not at top of page)
..... . . .....v.... ... 36.0User moved cursor up (currently at top of page)
..... . . ......v... ... 37.0User moved cursor down
..... . . .......v.. ... 38.0User finished
..... . . ........v. ... 39.0User entry invalid - try again
..... . . .........v ... 40.0Next user action moves curser down
                         0.00
..... . . .......... v.. 41.0User Exits program
..... . . .......... .v. 42.0Next user action moves cursor up
..... . . .......... ..v 43.0Next user action invalid - try again
                         0.00
..... . . .d........ ... 2.00Check sub options for current current Primary
.d.d. . . .......... ... 4.00Continue testing for selected sub options
ss... . . .......... ... 8.00Is the cursor off screen?
d.ls. . . .......... ... 9.00Reblock screen
....s s s .......... ... 10.0Set screen variables
....d d d ss...d..dd ddd 11.0Begin editing process
..... . . d.ls...... ... 12.0Wait for user input
..... . . ...dssss.. ... 13.0Test user input
..... . . ....d....s sss 14.0Move cursor up
..... . . ......d... ... 15.0Move cursor down
..... . . .......d.. ... 23.0User quits - update variables
                         0.00
....7 7 . .......... .46 88.0 sele 10
....9 9 . ....1..... .79 90.0 sele 1
..... . . .......... ...  121 APPEND BLANK
..... . . .......... ...  121 append blank
..... . . .......... ...  122 REPLACE REC_NO WITH RECNO()
..... . . .......... ...  122 replace rec_no with recno()
..... . . .......... ...  123 REPLACE REC_TYPE WITH "T1"
..... . . .......... ...  124 REPLACE FILE_SEL WITH "SELE 1"
..... . . .......... ...  125 REPLACE FILE_REC_N WITH TREC
..... . . .......... ...  132 replace id with nextid
..... . . .......... ...  135 replace field_type with unow
```

Figure 5.10 Selected Dataflow Elements: Modification to the Database in Clusters 8-14

List all parts related to database commands: Replace, Append, Select in clusters 15-24.

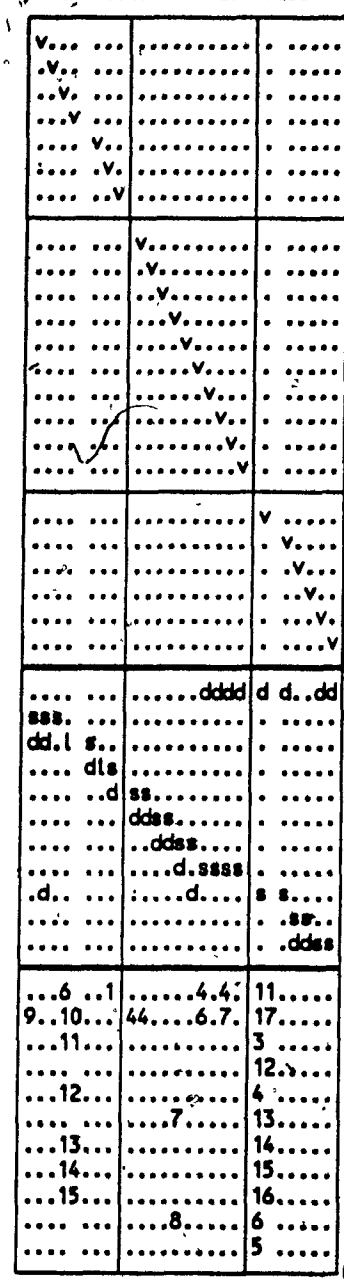


Figure 5.11 Selected Dataflow Elements: Modification to the Database in Clusters 15-2

5.3 ERRORS AND VERIFICATION

Program verification and error detection can take many forms. Testing for completeness, consistency and redundancy (described in chapter 4) is technically fairly easy for programs in table format.

The testing for completeness is basically a test for all possible permutations given a specific number of conditions. Given n conditions, there will be 2**n possible permutations. Take for example a cluster with 2 condition statements. There will be 2**2 or 4 possible permutations:

	conditions:	
	cond1	cond2
1.	t	t
2.	t	f
3.	f	t
4.	f	f

Figure 5.12 2**n Permutations Where n=2

Consider the following example taken from one of the test programs:

```
Program name: qmenuy
CLUSTER # 1   Starting Alternative # 1
|ss.....|.....| 1.0 Check each Primary Option selected
|tf.....|.....| 1.  tn<=6
```

Figure 5.13 Cluster 1, 1 Guard

```
Program name: qmenuy
CLUSTER # 2   Starting Alternative # 3
|d.sss...../|.........| 2.0  Check sub options for current current Primary
|tftt...../|.........| 1.   tn<=6
|..fft...../|.........| 2.   inTn=.T.

INCOMPLETE STEPS
Record#  COND1 COND2  CLUSTER L1 L2 L3
     4   F     F            2  0  0  0

AMBIGUOUS STEPS
Record#  COND1 COND2  CLUSTER L1 L2 L3
     3   T     F            2  3  4  0
```

Figure 5.14 Cluster 2, 2 Guards

Analysis of Cluster 2 indicates one step was not included, namely, for (.not.tn<=6) .and. (.not. inTn=.T.). This may indicate an error, or a mutually exclusive situation to be discussed further in the next example. There was also an ambiguous step found. Note that both alternative 3 and alternative 4 are the same. This was probably a typographical error.

The next example indicates what can happen on analysis if mutual exclusion of the predicates themselves is not considered (referred to earlier as "Intra-rule Inconsistency). In this example "&tnow" can only have one string value, namely: 1) "t0" or 2) "t1" or 3) "t2" or 4) "t3" or 5) "t4" or 6) "t5" or 7) "t6" or 8) anything else. Because mutual exclusion was not considered originally, the result of the analysis produced a massive result. Note that 2**8 is 256 possible combinations. The result is shown in Figure 5.15.

CLUSTER # 5

.....tf	fffff....	19.0 tnow="t0"
.....ft	fffff....	20.0 tnow="t1"
.....ff	tffff....	31.0 tnow="t2"
.....ff	ftfff....	32.0 tnow="t3"
.....ff	fftff....	33.0 tnow="t4"
.....ff	fftff....	34.0 tnow="t5"
.....ff	fftf....	35.0 tnow="t6"
.....	.....t....	30.0 otherwise

INCOMPLETE STEPS

Record#	COND1	COND2	COND3	COND4	COND5	COND6	COND7	COND8	CLUSTER	L1	L2	L3
1	T	T	T	T	T	T	T	T	1	0	0	0
2	F	T	T	T	T	T	T	T	1	0	0	0
3	T	F	T	T	T	T	T	T	2	0	0	0
4	F	F	T	T	T	T	T	T	2	0	0	0
5	T	T	F	T	T	T	T	T	1	0	0	0
6	F	T	F	T	T	T	T	T	2	0	0	0
7	T	F	F	T	T	T	T	T	1	0	0	0
8	F	F	F	T	T	T	T	T	2	0	0	0
9	T	T	T	F	T	T	T	T	2	0	0	0
10	F	T	T	F	T	T	T	T	1	0	0	0
11	T	F	T	F	T	T	T	T	1	0	0	0
12	F	F	T	F	T	T	T	T	1	0	0	0
13	T	T	F	F	T	T	T	T	1	0	0	0
14	F	T	F	F	T	T	T	T	1	0	0	0
15	T	F	F	F	T	T	T	T	1	0	0	0
16	F	F	F	F	T	T	T	T	1	0	0	0
17	T	T	T	T	F	T	T	T	0	0	0	0
18	F	T	T	T	F	T	T	T	0	0	0	0
19	T	F	T	T	F	T	T	T	0	0	0	0
20	F	F	T	T	F	T	T	T	0	0	0	0
21	T	T	F	T	F	T	T	T	0	0	0	0
22	F	T	F	T	F	T	T	T	0	0	0	0
23	T	F	F	T	F	T	T	T	0	0	0	0
24	F	F	F	T	F	T	T	T	0	0	0	0
25	T	T	T	F	F	T	T	T	0	0	0	0
26	F	T	T	F	F	T	T	T	0	0	0	0
27	T	F	T	F	F	T	T	T	0	0	0	0
28	F	F	T	F	F	T	T	T	0	0	0	0
29	T	T	F	F	F	T	T	T	0	0	0	0
30	F	T	F	F	F	T	T	T	0	0	0	0
31	T	F	F	F	F	T	T	T	0	0	0	0
32	F	F	F	F	F	T	T	T	0	0	0	0
33	T	T	T	T	T	F	T	T	0	0	0	0
34	F	T	T	T	T	F	T	T	0	0	0	0
35	T	F	T	T	T	F	T	T	0	0	0	0
36	F	F	T	T	T	F	T	T	0	0	0	0
37	T	T	F	T	T	F	T	T	0	0	0	0
38	F	T	F	T	T	F	T	T	0	0	0	0
39	T	F	F	T	T	F	T	T	0	0	0	0
40	F	F	F	T	T	F	T	T	0	0	0	0
41	T	T	T	F	T	F	T	T	0	0	0	0
42	F	T	T	F	T	F	T	T	0	0	0	0
43	T	F	T	F	T	F	T	T	0	0	0	0
44	F	F	T	F	T	F	T	T	0	0	0	0
45	T	T	F	F	T	F	T	T	0	0	0	0
46	F	T	F	F	T	F	T	T	0	0	0	0
47	T	F	F	F	T	F	T	T	0	0	0	0
48	F	F	F	F	T	F	T	T	0	0	0	0
49	T	T	T	T	F	F	T	T	0	0	0	0
50	F	T	T	T	F	F	T	T	0	0	0	0
51	T	F	T	T	F	F	T	T	0	0	0	0
52	F	F	T	T	F	F	T	T	0	0	0	0
53	T	T	F	T	F	F	T	T	0	0	0	0
54	F	T	F	T	F	F	T	T	0	0	0	0
55	T	F	F	T	F	F	T	T	0	0	0	0
56	F	F	F	T	F	F	T	T	0	0	0	0
57	T	T	T	F	F	F	T	T	0	0	0	0
58	F	T	T	F	F	F	T	T	0	0	0	0

59	T	F	T	F	F	F	T	T	0	0	0	0
60	F	F	T	F	F	F	T	T	0	0	0	0
61	T	T	F	F	F	F	T	T	0	0	0	0
62	F	T	F	F	F	F	T	T	0	0	0	0
63	T	F	F	F	F	F	T	T	0	0	0	0
65	T	T	T	T	T	T	F	T	1	0	0	0
66	F	T	T	T	T	T	F	T	1	0	0	0
67	T	F	T	T	T	T	F	T	2	0	0	0
68	F	F	T	T	T	T	F	T	2	0	0	0
69	T	T	F	T	T	T	F	T	1	0	0	0
70	F	T	F	T	T	T	F	T	2	0	0	0
71	T	F	F	T	T	T	F	T	1	0	0	0
72	F	F	F	T	T	T	F	T	2	0	0	0
73	T	T	T	F	T	T	F	T	2	0	0	0
74	F	T	T	F	T	T	F	T	1	0	0	0
75	T	F	T	F	T	T	F	T	1	0	0	0
76	F	F	T	F	T	T	F	T	1	0	0	0
77	T	T	F	F	T	T	F	T	1	0	0	0
78	F	T	F	F	T	T	F	T	1	0	0	0
79	T	F	F	F	T	T	F	T	1	0	0	0
80	F	F	F	F	T	T	F	T	1	0	0	0
81	T	T	T	T	F	T	F	T	0	0	0	0
82	F	T	T	T	F	T	F	T	0	0	0	0
83	T	F	T	T	F	T	F	T	0	0	0	0
84	F	F	T	T	F	T	F	T	0	0	0	0
85	T	T	F	T	F	T	F	T	0	0	0	0
86	F	T	F	T	F	T	F	T	0	0	0	0
87	T	F	F	T	F	T	F	T	0	0	0	0
88	F	F	F	T	F	T	F	T	0	0	0	0
89	T	T	T	F	F	T	F	T	0	0	0	0
90	F	T	T	F	F	T	F	T	0	0	0	0
91	T	F	T	F	F	T	F	T	0	0	0	0
92	F	F	T	F	F	T	F	T	0	0	0	0
93	T	T	F	F	F	T	F	T	0	0	0	0
94	F	T	F	F	F	T	F	T	0	0	0	0
95	T	F	F	F	F	T	F	T	0	0	0	0
97	T	T	T	T	T	F	F	T	0	0	0	0
98	F	T	T	T	T	F	F	T	0	0	0	0
99	T	F	T	T	T	F	F	T	0	0	0	0
100	F	F	T	T	T	F	F	T	0	0	0	0
101	T	T	F	T	T	F	F	T	0	0	0	0
102	F	T	F	T	T	F	F	T	0	0	0	0
103	T	F	F	T	T	F	F	T	0	0	0	0
104	F	F	F	T	T	F	F	T	0	0	0	0
105	T	T	T	F	T	F	F	T	0	0	0	0
106	F	T	T	F	T	F	F	T	0	0	0	0
107	T	F	T	F	T	F	F	T	0	0	0	0
108	F	F	T	F	T	F	F	T	0	0	0	0
109	T	T	F	F	T	F	F	T	0	0	0	0
110	F	T	F	F	T	F	F	T	0	0	0	0
111	T	F	F	F	T	F	F	T	0	0	0	0
113	T	T	T	T	F	F	F	T	0	0	0	0
114	F	T	T	T	F	F	F	T	0	0	0	0
115	T	F	T	T	F	F	F	T	0	0	0	0
116	F	F	T	T	F	F	F	T	0	0	0	0
117	T	T	F	T	F	F	F	T	0	0	0	0
118	F	T	F	T	F	F	F	T	0	0	0	0
119	T	F	F	T	F	F	F	T	0	0	0	0
121	T	T	T	F	F	F	F	T	0	0	0	0
122	F	T	T	F	F	F	F	T	0	0	0	0
123	T	F	T	F	F	F	F	T	0	0	0	0
125	T	T	F	F	F	F	F	T	0	0	0	0
129	T	T	T	T	T	T	T	F	1	0	0	0
130	F	T	T	T	T	T	T	F	1	0	0	0
131	T	F	T	T	T	T	T	F	2	0	0	0
132	F	F	T	T	T	T	T	F	2	0	0	0
133	T	T	F	T	T	T	T	F	1	0	0	0
134	F	T	F	T	T	T	T	F	2	0	0	0
135	T	F	F	T	T	T	T	F	1	0	0	0
136	F	F	F	T	T	T	T	F	2	0	0	0

137	T	T	T	F	T	T	T	F	2	0	0	0
138	F	T	T	F	T	T	T	F	1	0	0	0
139	T	F	T	F	T	T	T	F	1	0	0	0
140	F	F	T	F	T	T	T	F	1	0	0	0
141	T	T	F	F	T	T	T	F	1	0	0	0
142	F	T	F	F	T	T	T	F	1	0	0	0
143	T	F	F	F	T	T	T	F	1	0	0	0
144	F	F	F	F	T	T	T	F	1	0	0	0
145	T	T	T	T	F	T	T	F	0	0	0	0
146	F	T	T	T	F	T	T	F	0	0	0	0
147	T	F	T	T	F	T	T	F	0	0	0	0
148	F	F	T	T	F	T	T	F	0	0	0	0
149	T	T	F	T	F	T	T	F	0	0	0	0
150	F	T	F	T	F	T	T	F	0	0	0	0
151	T	F	F	T	F	T	T	F	0	0	0	0
152	F	F	F	T	F	T	T	F	0	0	0	0
153	T	T	T	F	F	T	T	F	0	0	0	0
154	F	T	T	F	F	T	T	F	0	0	0	0
155	T	F	T	F	F	T	T	F	0	0	0	0
156	F	F	T	F	F	T	T	F	0	0	0	0
157	T	T	F	F	F	T	T	F	0	0	0	0
158	F	T	F	F	F	T	T	F	0	0	0	0
159	T	F	F	F	F	T	T	F	0	0	0	0
160	F	F	F	F	F	T	T	F	0	0	0	0
161	T	T	T	T	T	F	T	F	0	0	0	0
162	F	T	T	T	T	F	T	F	0	0	0	0
163	T	F	T	T	T	F	T	F	0	0	0	0
164	F	F	T	T	T	F	T	F	0	0	0	0
165	T	T	F	T	T	F	T	F	0	0	0	0
166	F	T	F	T	T	F	T	F	0	0	0	0
167	T	F	F	T	T	F	T	F	0	0	0	0
168	F	F	F	T	T	F	T	F	0	0	0	0
169	T	T	T	F	T	F	T	F	0	0	0	0
170	F	T	T	F	T	F	T	F	0	0	0	0
171	T	F	T	F	T	F	T	F	0	0	0	0
172	F	F	T	F	T	F	T	F	0	0	0	0
173	T	T	F	F	T	F	T	F	0	0	0	0
174	F	T	F	F	T	F	T	F	0	0	0	0
175	T	F	F	F	T	F	T	F	0	0	0	0
176	F	F	F	F	T	F	T	F	0	0	0	0
177	T	T	T	T	F	F	T	F	0	0	0	0
178	F	T	T	T	F	F	T	F	0	0	0	0
179	T	F	T	T	F	F	T	F	0	0	0	0
180	F	F	T	T	F	F	T	F	0	0	0	0
181	T	T	F	T	F	F	T	F	0	0	0	0
182	F	T	F	T	F	F	T	F	0	0	0	0
183	T	F	F	T	F	F	T	F	0	0	0	0
184	F	F	F	T	F	F	T	F	0	0	0	0
185	T	T	T	F	F	F	T	F	0	0	0	0
186	F	T	T	F	F	F	T	F	0	0	0	0
187	T	F	T	F	F	F	T	F	0	0	0	0
188	F	F	T	F	F	F	T	F	0	0	0	0
189	T	T	F	F	F	F	T	F	0	0	0	0
190	F	T	F	F	F	F	T	F	0	0	0	0
191	T	F	F	F	F	F	T	F	0	0	0	0
193	T	T	T	T	T	T	F	F	1	0	0	0
194	F	T	T	T	T	T	F	F	1	0	0	0
195	T	F	T	T	T	T	F	F	2	0	0	0
196	F	F	T	T	T	T	F	F	2	0	0	0
197	T	T	F	T	T	T	F	F	1	0	0	0
198	F	T	F	T	T	T	F	F	2	0	0	0
199	T	F	F	T	T	T	F	F	1	0	0	0
200	F	F	F	T	T	T	F	F	2	0	0	0
201	T	T	T	F	T	T	F	F	2	0	0	0
202	F	T	T	F	T	T	F	F	1	0	0	0
203	T	F	T	F	T	T	F	F	1	0	0	0
204	F	F	T	F	T	T	F	F	1	0	0	0
205	T	T	F	F	T	T	F	F	1	0	0	0
206	F	T	F	F	T	T	F	F	1	0	0	0
207	T	F	F	F	T	T	F	F	1	0	0	0

208	F	F	F	F	T	T	F	F	1	0	0	0
209	T	T	T	T	F	T	F	F	0	0	0	0
210	F	T	T	T	F	T	F	F	0	0	0	0
211	T	F	T	T	F	T	F	F	0	0	0	0
212	F	F	T	T	F	T	F	F	0	0	0	0
213	T	T	F	T	F	T	F	F	0	0	0	0
214	F	T	F	T	F	T	F	F	0	0	0	0
215	T	F	F	T	F	T	F	F	0	0	0	0
216	F	F	F	T	F	T	F	F	0	0	0	0
217	T	T	T	F	F	T	F	F	0	0	0	0
218	F	T	T	F	F	T	F	F	0	0	0	0
219	T	F	T	F	F	T	F	F	0	0	0	0
220	F	F	T	F	F	T	F	F	0	0	0	0
221	T	T	F	F	F	T	F	F	0	0	0	0
222	F	T	F	F	F	T	F	F	0	0	0	0
223	T	F	F	F	F	T	F	F	0	0	0	0
225	T	T	T	T	T	F	F	F	0	0	0	0
226	F	T	T	T	T	F	F	F	0	0	0	0
227	T	F	T	T	T	F	F	F	0	0	0	0
228	F	F	T	T	T	F	F	F	0	0	0	0
229	T	T	F	T	T	F	F	F	0	0	0	0
230	F	T	F	T	T	F	F	F	0	0	0	0
231	T	F	F	T	T	F	F	F	0	0	0	0
232	F	F	F	T	T	F	F	F	0	0	0	0
233	T	T	T	F	T	F	F	F	0	0	0	0
234	F	T	T	F	T	F	F	F	0	0	0	0
235	T	F	T	F	T	F	F	F	0	0	0	0
236	F	F	T	F	T	F	F	F	0	0	0	0
237	T	T	F	F	T	F	F	F	0	0	0	0
238	F	T	F	F	T	F	F	F	0	0	0	0
239	T	F	F	F	T	F	F	F	0	0	0	0
241	T	T	T	T	F	F	F	F	0	0	0	0
242	F	T	T	T	F	F	F	F	0	0	0	0
243	T	F	T	T	F	F	F	F	0	0	0	0
244	F	F	T	T	F	F	F	F	0	0	0	0
245	T	T	F	T	F	F	F	F	0	0	0	0
246	F	T	F	T	F	F	F	F	0	0	0	0
247	T	F	F	T	F	F	F	F	0	0	0	0
249	T	T	T	F	F	F	F	F	0	0	0	0
250	F	T	T	F	F	F	F	F	0	0	0	0
251	T	F	T	F	F	F	F	F	0	0	0	0
253	T	T	F	F	F	F	F	F	0	0	0	0
256	F	F	F	F	F	F	F	F	0	0	0	0

Figure 5.15 Analysis of cluster where mutual exclusion of Predicates is not considered

```
CLUSTER #      5
|........tf|ffffff....| 19.0 tnow="t0"
|........ft|ffffff....| 20.0 tnow="t1"
|........ff|tffffff....| 31.0 tnow="t2"
|........ff|ftffff....| 32.0 tnow="t3"
|........ff|fftfff....| 33.0 tnow="t4"
|........ff|ffftff....| 34.0 tnow="t5"
|........ff|fffftf....| 35.0 tnow="t6"
|..........|.....t....| 30.0 otherwise
INCOMPLETE STEPS
Record#  COND1 COND2 COND3 COND4 COND5 COND6 COND7 COND8 CLUSTER L1 L2 L3
   256 F     F     F     F     F     F     F     F           0  0  0  0
```

Figure 5.16 Analysis of cluster where mutual exclusion of Predicates is not considered

If mutual exclusion is considered in this example, then no two conditions can be true at the same time. The result of taking into account mutual exclusion is shown in Figure 5.16. Here, the only incomplete step is when all conditions are false. This too is not possible because of the "otherwise" condition.

Another example of a mutual exclusion example is shown in Figure 5.17. Conditions 4 and 5 are mutually exclusive. However, analysis shows ambiguous states rather than incomplete states. Again, if mutual exclusion of the predicates is considered, then states 1 and 2 in the list of "AMBIGUOUS STEPS" would be eliminated. States 3 and 5 would need definition in the program, or an indication of exclusion with steps 4 and/or 5.

Analysis is useful for programs having large numbers of alternatives. It is a handy check that tends to indicate typographical errors and relationships between conditions. The program QANALY1 does this analysis.

```
Program name: qmafuy
CLUSTER # 3  Starting Alternative # 6
|.dd.dsss.|.........|  3.0 Test if sub Option selected
|.....t..f.|.........|  3.  &tnow="V" .or. &tnow="V"
|......t.f.|.........|  4.  i>0 .and. i<10
|.......tf.|.........|  5.  i>9

AMBIGUOUS STEPS
Record#  COND1 COND2 COND3  CLUSTER L1 L2 L3
     1     T     T     T          3  6  7  8
     2     F     T     T          3  7  8  0
     3     T     F     T          3  6  8  0
     5     T     T     F          3  6  7  0

Record#  COND1 COND2 COND3  CLUSTER L1 L2 L3
     1     T     T     T          3  6  7  8
```

Figure 5.17 Cluster 3, starting Alternative 6: Example of Mutual Exclusion of Guards

5.4 PROGRAM GENERATOR

A program generator has been produced for dBase III. The input to the generator is database information. This information is seen in Figure 5.2. The generator is normally used in the final implementation process of a program. Several benchmarks were made on programs produced in the normal way versus the generated programs as described here. The efficiency of the generated programs is somewhat slower than equivalent programs written in the more conventional way. However, the speed differences are not normally significant. Occasionally a few changes to a generated subroutine may be required to improve the speed. These programs tend to be longer.

The dBASE program generator produced the program in

							0.08
........V.	....	....	....	....	....	. .....	9.00t0: initialize flags
.........V	....	....	....	....	....	. .....	10.0t1:option selected
....	V.........	....	....	....	....	. .....	11.0t2:option selected
....	.V........	....	....	....	....	. .....	12.0t3:option selected
....	..V.......	....	....	....	....	. .....	13.0t4:option selected
....	...V......	....	....	....	....	. .....	14.0t5:option selected
....	....V.....	....	....	....	....	. .....	15.0t6:option selected
....	.....V....	....	....	....	....	. .....	16.0None of the tn options selected
....	......V...	....	....	....	....	. .....	17.0
....	.......V..	....	....	....	....	. .....	18.0
....	........V.	....	....	....	....	. .....	19.0
....	V	....	....	....	....	. .....	20.0
....	....	V....... . .	....	....	....	. .....	21.0
....	....	.V...... . .	....	....	....	. .....	22.0
....	....	..V..... . .	....	....	....	. .....	23.0
....	....	...V.... . .	....	....	....	. .....	24.0Current Position off screen; reblock
....	....	....V... . .	....	....	....	. .....	25.0Reblocking screen not necessary
....	....	.....V.. . .	....	....	....	. .....	26.0Display current screen block
....	....	......V. . .	....	....	....	. .....	27.0Screen block complete-close block at
							0.00
....d.ss..	....	....d.d. . .	....	....	....	. .....	4.00Continue testing for selected sub op
......d.ss	ssssss....	....	....	....	....	. .....	5.00Test if Primary Option Selected
.........d	......ss..	....	....	....	....	. .....	6.00Are there more sub options to test f
....	......d.ss	sss..... . .	....	....	....	. .....	7.00Test current Sub Option
........d.	ddddd*.ddd	dddss... . .	....	....	....	. .....	8.00Is the cursor off screen?
....	....	...d.ls. . .	....	....	....	. .....	9.00Reblock screen
.......d..	....	.......s s s	....	....	....	. .....	10.0Set screen variables
							0.00
........tf	ffffff....	....	....	....	....	. .....	19.0 tnow="t0"
........ft	ffffff....	....	....	....	....	. .....	20.0 tnow="t1"
....	......tf..	....	....	....	....	. .....	21.0 un <= 4
....	........tf	fff..... . .	....	....	....	. .....	22.0 un=1
....	........t.	....	....	....	....	. .....	23.0 u1="v" .or. u1="V"
....	........ft	fff..... . .	....	....	....	. .....	24.0 un=2
....	.........t	....	....	....	....	. .....	25.0 u2="v" .or. u2="V"
....	........ff	tff..... . .	....	....	....	. .....	26.0 un=3
....	....	t....... . .	....	....	....	. .....	27.0 u3="v" .or. u3="V"
....	........ff	ftf..... . .	....	....	....	. .....	28.0 un=4
....	....	.t...... . .	....	....	....	. .....	29.0 u4="v" .or. u4="V"
....	.....t..ff	fft..... . .	....ffftf	fft.... ...	......ffft	. .....	30.0 otherwise
........ff	tfffff....	....	....	....	....	. .....	31.0 tnow="t2"
........ff	ftffff....	....	....	....	....	. .....	32.0 tnow="t3"
........ff	fftfff....	....	....	....	....	. .....	33.0 tnow="t4"
........ff	ffftff....	....	....	....	....	. .....	34.0 tnow="t5"
........ff	fffftf....	....	....	....	....	. .....	35.0 tnow="t6"
....	....	...tf... . .	....	....	....	. .....	36.0 reblock
....	....	.....tf. . .	....	....	....	. .....	37.0 tvar<=23
							0.00===
....	....	......3. . .	....	....	....	. .....	22.0 toff2=1
....	....	......4. . .	....	....	....	. .....	25.0 nextid=0
........1.	.......1..	....	....	....	....	. .....	23.0 tcol=4
....	....	......6. . .	....	....	....	. .....	36.0 reblock=.f.
2.......5.	.......5..	....	....	....	....	. .....	43.0 reblock=.t.
....1...4.	.......4..	....	....	....	....	. .....	46.0 tnowcont=.t.
.....1..2.	....	....	....	....	....	. .....	49.0 tn=tn+1
.....2..3.	.......3..	....	....	....	....	. .....	50.0 tnow="t"+ltrim(str(tn))
.........1	....	....	....	....	....	. .....	51.0 @ trow,tcol say "v"
.........2	....	....	....	....	....	. .....	52.0 @ arow,tcol say "v"
.........3	....	....	....	....	....	. .....	53.0 @ 4+tot1,2 say chr(195)+substr(sstr
.........4	....	....	....	....	....	. .....	54.0 @ trow,2*tspace+7 say "1. IDEA Deve
....	......12..	....	....	....	....	. .....	55.0 un=un+1
....	......2...	....	....	....	....	. .....	56.0 toff3=tot1 + 4
....	......3.33	33...... . .	....	....	....	. .....	57.0 tnowcont=.f.
....	......4...	....	....	....	....	. .....	58.0 unow="u"+ltrim(str(un))
....	........1.	....	....	....	....	. .....	59.0 @ arow,2*tspace+7 say "1. Determine
....	22	22...... . .	....	....	....	. .....	60.0 arow=arow+1
....	........4.	....	....	....	....	. .....	61.0 set filter to field_type="U1"
....	1	....	....	....	....	. .....	62.0 @ arow,2*tspace+7 say "2. Document
....	4	....	....	....	....	. .....	63.0 set filter to field_type="U2"
....	....	1....... . .	....	....	....	. .....	64.0 @ arow,2*tspace+7 say "3. Partition
....	....	4....... . .	....	....	....	. .....	65.0 set filter to field_type="U3"
....	....	.1...... . .	....	....	....	. .....	66.0 @ arow,2*tspace+7 say "4. Determine
....	....	.4...... . .	....	....	....	. .....	67.0 set filter to field_type="U4"

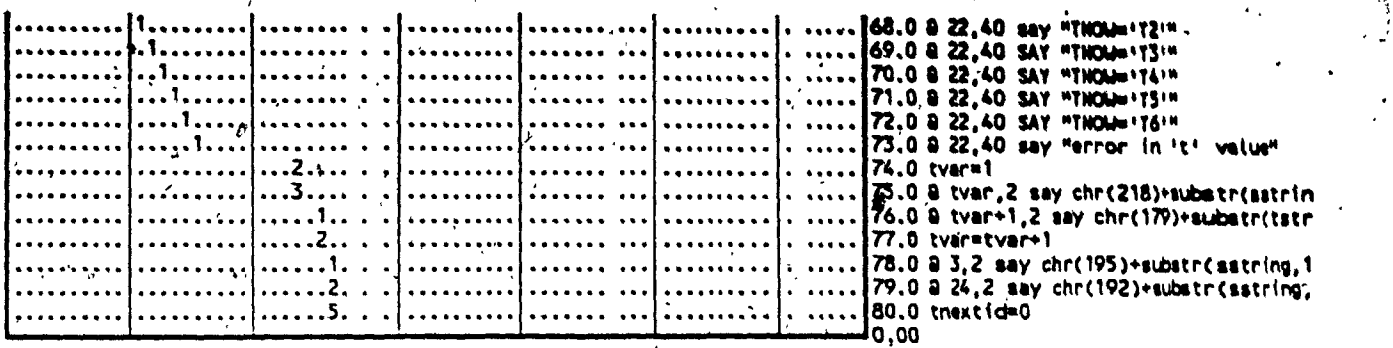


Figure 5.18 Computer Generated Table Representation Program "ABC"

Figure 5.19. This is the same program represented in table form in Figure 5.2. This generated program is highly structured with documentation imbedded in the program. This reiterates the point that documentation is always relevant to the program even though modifications have been made.

The same program written in the conventional manner is given in Figure 5.20. A comparison of style, structure, documentation, and the ability to review makes the generated program much more desirable.

The major weakness of the generated program is the time it takes to generate it. This is due to the slow response time during searches made to database for information. Currently it generates the whole program, not just the modified parts. However, enhancements to the generator could make the editing process more efficient. Due to the generated program structure, generation of a single alternative could be edited back into the original generated program fairly quickly. Also, a database management system capable of faster searches as well as faster hardware could

```
                PROGRAM PRODUCED BY GENERATOR
CLUSTER=1
EXIT=.f.
DO WHILE .NOT. EXIT
   DO CASE
      :
      :
      *5.0Test if Primary Option Selected
      CASE CLUSTER=        5
           DO CASE
                **5.1t0: initialize flags
                CASE tnow="t0".and..not. case tnow="t1".and..not.tnow="t2".and..not.tnow="t3"
                     .and..not.tnow="t4".and..not.tnow="t5".and..not.tnow="t6"
                     tcol=4
                     tn=tn+1
                     tnow="t"+ltrim(str(tn))
                     tnowcont=.t.
                     reblock=.t.
                    CLUSTER=         8
                **5.2t1:option selected
                CASE .not.tnow="t0".and. case tnow="t1".and..not.tnow="t2".and..not.tnow="t3"
                     .and..not.tnow="t4" .and..not.tnow="t5".and..not.tnow="t6"
                     @ trow,tcol say "v"
                     @ arow,tcol say "v"
                     @ 4+tot1,2 say chr(195)+substr(sstring,1,2*tspace+1)+chr(180)
                     @ trow,2*tspace+7 say "1. IDEA Development Features"
                    CLUSTER=         6
                **5.3t2:option selected
                CASE .not.tnow="t0".and..not. case now="t1".and.tnow="t2".and..not.tnow="t3"
                     .and..not.tnow="t4".and..not.tnow="t5".and..not.tnow="t6"
                     @ 22,40 say "TNOW='T2'"
                    CLUSTER=         8
                **5.4t3:option selected
                CASE .not.tnow="t0".and..not. case now="t1".and..not.tnow="t2".and.tnow="t3"
                     .and..not.tnow="t4".and..not.tnow="t5".and..not.tnow="t6"
                     @ 22,40 SAY "TNOW='T3'"
                    CLUSTER=         8
                **5.5t4:option selected
                CASE .not.tnow="t0".and..not. case now="t1".and..not.tnow="t2".and..not.tnow="t3"
                     .and.tnow="t4".and..not.tnow="t5".and..not.tnow="t6"
                     @ 22,40 SAY "TNOW='T4'"
                    CLUSTER=         8
                **5.6t5:option selected
                CASE .not.tnow="t0".and..not. case now="t1".and..not.tnow="t2".and..not.tnow="t3"
                     .and..not.tnow="t4".and.tnow="t5".and..not.tnow="t6"
                     @ 22,40 SAY "TNOW='T5'"
                    CLUSTER=         8
                **5.7t6:option selected
                CASE .not.tnow="t0".and..not. case tnow="t1".and..not.tnow="t2".and..not.tnow="t3"
                     .and..not.tnow="t4".and..not.tnow="t5".and.tnow="t6"
                     @ 22,40 SAY "TNOW='T6'"
                    CLUSTER=         8
                **5.8None of the tn options selected
                OTHERWISE
                     @ 22,40 say "error in 't' value"
                    CLUSTER=         8
           ENDCASE
      *6.0Are there more sub options to test for?
      CASE CLUSTER=        6
           DO CASE
                **6.1
                CASE  if un <= 4
                     un=un+1
                     toff3=tot1 + 4
                     tnowcont=.f.
                     unow="u"+ltrim(str(un))
                    CLUSTER=         7
```

```
          **6.2
          CASE .not. if un <= 4
               tcol=4
               un=un+1
               tnow="t"+ltrim(str(tn))
               tnowcont=.t.
               reblock=.t.
              CLUSTER=          8
     ENDCASE

*7.0Test current Sub Option
CASE CLUSTER=          7
     DO CASE
          **7.1
          CASE  case un=1.and. if u1="v" .or. u1="V".and..not.un=2.and..not.un=3.and..not.un=4
               @ arow,2*tspace+7 say "1. Determine System Requirements"
               arow=arow+1
               tnowcont=.f.
               set filter to field_type="U1"
              CLUSTER=          8
          **7.2
          CASE .not. case un=1.and.un=2.and.u2="v" .or. u2="V".and..not.un=3.and..not.un=4
               @ arow,2*tspace+7 say "2. Document Users Needs"
               arow=arow+1
               tnowcont=.f.
               set filter to field_type="U2"
              CLUSTER=          8
          **7.3
          CASE .not. case un=1.and..not.un=2.and.un=3.and.u3="v" .or. u3="V".and..not.un=4
               @ arow,2*tspace+7 say "3. Partition/Allocate functional requirements"
               arow=arow+1
               tnowcont=.f.
               set filter to field_type="U3"
              CLUSTER=          8
          **7.4
          CASE .not. case un=1.and..not.un=2.and..not.un=3.and.un=4.and.u4="v" .or. u4="V"
               @ arow,2*tspace+7 say "4. Determine System Requirements"
               arow=arow+1
               tnowcont=.f.
               set filter to field_type="U4"
              CLUSTER=          8
          **7.5
          OTHERWISE
              CLUSTER=          8
     ENDCASE
*8.0Is the cursor off screen?
CASE CLUSTER=          8
     DO CASE
          **8.1Current Position off screen; reblock screen
          CASE reblock
               clear
               tvar=1
               @tvar,2 say chr(218)+substr(sstring,1,2*tspace+1)+chr(191)
               CLUSTER=          9
          CASE .not. reblock
               CLUSTER=          4
     ENDCASE
*9.0Reblock screen
CASE CLUSTER=          8
     DO CASE
          **9.1Display current screen block
          CASE tvar<=23
               @ tvar+1,2 say chr(179)+substr(tstring,1,2*tspace+1)+chr(179)
               tvar=tvar+1
               CLUSTER=          9
          **9.2Screen block complete-close block
          CASE .not. tvar<=23
               @ 3,2 say chr(195)+substr(sstring,1,2*tspace+1)+chr(180)
               @ 24,2 say chr(192)+substr(sstring,1,2*tspace+1)+chr(217)
               toff2=1
```

```
                nextid=0
                tnextid=1
                reblock=.f.
               CLUSTER=          4
        ENDCASE

    :
    :
 ENDCASE
ENDDO
```

Figure 5.19: Generated Program for "ABC"

speed the process considerably. The program in Figure 5.18 took about 12 minutes to produce on the IBM 4.7MHz machine. A 20MHz machine would probably reduce that time to about 4 minutes. The new dBase IV would probably reduce the time required even more.

Additional enhancements to the generator could be the verification of the program during processing. However, the most common types of errors (completeness of the guard) are easily detected in the table format.

See Appendix D for the complete programs given as examples in this chapter.

CONVENTIONAL CODING

```
****QMENU6.PRG
    :
    :
* 4. Select next sub-item for current Tn
*    4.1  Sub-item selected: print list of sub-item descriptions;
*         read first item in list
*         --->5
*    4.2  Sub-item not selected; get next sub-item
*         --->4
*    4.3  Sub-item now out of range: get next Tn
*         --->4
*    4.4  No more items or sub-items remain to be printed
*         --->0
    :
    :
PUBLIC TOFF,TCOL,TROW,NEXTID,TN,UN,VN,WN
public XN,YN,ZN,TNOWCONT,TNOW,UNOW
:
:
**4.0     ****LOOP FOR WITHIN A GIVEN Tn
          do while tnowcont
     @ 23,25 SAY STR(NEXTID,4,0)+"    "+STR(TID,3,0)+STR(TNEXTID,4,0)+"   "+STR(TOFF,3,0)+" "+STR(TOFF2,4,0)+"  "+$
     ? STR(NEXTID,4,0)+"    "+STR(TID,3,0)+STR(TNEXTID,4,0)+"   "+STR(TOFF,3,0)+" "+STR(TOFF2,4,0)+"  "+STR(TSTART,
**4.1
               @ trow,2*tspace+7
               do case
               case tnow="t0"
                    tcol=4
                    tn=tn+1
                    tnow="t"+ltrim(str(tn))
                    tnowcont=.t.
                    reblock=.t.
               case tnow="t1"
                    @ trow,tcol say "v"
                    @ arow,tcol say "v"
                    @ 4+tot1,2 say chr(195)+substr(sstring,1,2*tspace+1)+chr(180)
                    @ trow,2*tspace+7 say "1. IDEA Development Features"
                    if un <= 4
                         un=un+1
                         toff3=tot1 + 4
                         tnowcont=.f.
                         unow="U"+ltrim(str(un))
                         do case
                         * DETERMINE WHICH "u" *
                              case un=1
                                   if u1="v" .or. u1="V"
                                        @ arow,2*tspace+7 say "1. Determine System Requirements"
                                        arow=arow+1
                                        tnowcont=.f.
                                        set filter to field_type="U1"
                                   endif
                              case un=2
                                   if u2="v" .or. u2="V"
                                        @ arow,2*tspace+7 say "2. Document Users Needs"
                                        arow=arow+1
                                        tnowcont=.f.
                                        set filter to field_type="U2"
                                   endif
                              case un=3
                                   if u3="v" .or. u3="V"
                                        @ arow,2*tspace+7 say "3. Partition/Allocate functional requirements"
                                        arow=arow+1
                                        tnowcont=.f.
                                        set filter to field_type="U3"
                                   endif
                              case un=4
```

```
                    if u4="y" .or. u4="Y"
                        @ arow,2*tspace+7 say "4. Determine System Requirements"
                        arow=arow+1
                        tnowcont=.f.
                        set filter to field_type="U4"
                    endif
                otherwise
                    ****do nothing; go-back and get  next Un
            endcase
        else
         *****GET NEXT Tn
            tcol=4
            tn=tn+1
            tnow="t"+ltrim(str(tn))
            tnowcont=.t.
            reblock=.t.
        endif
    case tnow="t2"
        @ 22,40 say "TNOW='T2'"
        tnowcont=.f.
    case tnow="t3"
        @ 22,40 SAY "TNOW='T3'"
        tnowcont=.f.
    case tnow="t4"
        @ 22,40 SAY "TNOW='T4'"
        tnowcont=.f.
    case tnow="t5"
        @ 22,40 SAY "TNOW='T5'"
        tnowcont=.f.
    case tnow="t6"
        @ 22,40 SAY "TNOW='T6'"
        tnowcont=.f.
    otherwise
        @ 22,40 say "error in 't' value"
    endcase
    if reblock
     *********BLOCK SCREEN***************
            clear
            tvar=1
            @ tvar,2 say chr(218)+substr(sstring,1,2*tspace+1)+chr(191)
            do while tvar<=23
                @ tvar+1,2 say chr(179)+substr(tstring,1,2*tspace+1)+chr(179)
                tvar=tvar+1
            enddo
            @ 3,2 say chr(195)+substr(sstring,1,2*tspace+1)+chr(180)
            @ 24,2 say chr(192)+substr(sstring,1,2*tspace+1)+chr(217)
            toff2=1
            nextid=0
            tnextid=1
            reblock=.f.
        endif
    enddo
    *while tnowcont
    tcontinue=.t.
    finish=.f.
    flagstart=.t.
    goto top
    tid=nextid
    i=24
    toff=tstart
    trow=2
    :
    :
  enddo
  ***inTn
enddo
** tn<=6
```

Figure 5.20: Conventional Program for "ABC"

5.5 CONCLUSIONS

The ability to analyze programs can provide additional information to produce a more error free final product than would normally be possible. Viewing a program graphically allows control flow to be reviewed quickly. The logical review of control flow can be done using the J-Map Table representation. Data flow can be reviewed using database selection processes.

The ability to review a program from different perspectives can be accomplished if the program is structured in a format that can be stored in a database. Other reviews and analyses can be done on programs in a way similar to the business use of databases. To design and implement a major system using these ideas may indeed provide reviews of information flow in programs that were never considered to be practical before. A good test of a system is applications of the system to real situations. This approach to programming could provide the manager/designer/programmer with a rich environment conducive to software design.

6.0 CONCLUDING REMARKS

The environment proposed in this thesis provides and supports a structured method of approach conducive to software design. The system framework is an organized database of information supporting all details known about the problem being solved and details on how it was solved. It becomes in effect the knowledge base used to solve a particular problem. Documentation is imbedded in the implementation process.

This knowledge base can be reviewed in a variety of ways as demonstrated in the thesis. Control flow tables, data flow review, graphical representation and automated verification are all supported using database query.

Automated analysis for completeness, consistency and redundancy was demonstrated. Database query was used to review data flow and control flow.

The ability to graphically represent a program provides an indication of complexity as well as control flow of the program. The partitioning process could well be reviewed with a "master" graphical overview of the problem.

The J-Map notation, based on sets, defined roles of sets and set members and associations between them, provides a complete methodology for partitioning large problems. The decomposition process may be performed with respect to time order, data flow, logical grouping, access to a common

resource, control flow or other criterion. This flexibility provides a gateway to other methodologies.

J-Maps can be applied to the design process as well as to the problem solution. It is also applicable to conversion processes and modelling of existing programs as described by Jaworski and Virand[37]. J-Maps are very versatile problem solving tools.

7.0 REFERENCES

1. H.G. Hughes, "A Programming Language Engineered for Beginners", Comput. Lang. Vol 10, No. 1, pp. 23-36, 1985.

2. Al Ries and Jack Trout, Positioning: The Battle For Your Mind, McGraw-Hill, Inc., 1981.

3. M.J. Meldmen, D.J. McLeod, R.J. Pellicore and M. Squire, RISS: A Relational Data Base Management System for Minicoumputers, Van Nostrand Reinhold Company, 1978.

4. dBASE III PLUS, Copyright Ashton-Tate, 1985 (As an unpublished work).

5. James Martin, System Design from Provably Correct Constructs, 1985, p. 30.

6. Martin, ibid, p. 31-32.

7. Gane and Sarson, Structured Systems Analysis, Prentice-Hall, 1979.

8. E. Yourdon and L. Constantine, STRUCTURED DESIGN: Fundamentals of a Discipline of Computer Program and Systems Design, Prentice-Hall, Inc., 1979.

9. Martin, ibid.

10. M. A. Jackson, Principles of Program Design, Academic Press, New York, 1975.

11. G.L. Bergland, "A Guided Tour of Program Design Methodologies", Computer, Volume 14, Number 10, October 1981, p. 18-37.

12. W.M. Jaworski, H. Hinterberger, "Controlled Program Design by Use of the Programming Concept ABL", Angewandte Informatik, July 1981, pp. 302-310.

13.W.M Jaworski, L Ficocelli, K.S. O'Mara, The ABL/W4 Methodology for System Modelling, System Research J.,Vol.4, No.1,(1987), pp 23-37.

14. R. Thurner, K. Bauknecht, "Procedural Decision Tables and Their Implementation", International Computing Symposium, 1973, p. 259-263.

15. G.L. Bergland, ibid.

16. E.W. Dijkstra, A Discipline of Programming, Prentice-Hall, Englewood Cliffs, N.J., 1976.

17. W.M. Jaworski, T. Radhakrishnan, Modelling of System Development Methodologies, CompInt '87 Proceedings, Conference on Computer Aided Technologies, November 9-12, 1987, Montreal, pp 1-5.

18. E. Yourdon and L. Constantine, Structured Design: Fundamentals of Discipline of Computer Program and Systems Design, Prentice-Hall, Inc, 1979.

19. IEEE Standard for Software Life Cycle Processes, Preliminary report, March 1, 1988.

20. Jaworski and Radhakrishnan, ibid.

21. N. Wirth, Systematic Programming, Prentice-Hall, Englewood Cliffs, N.J., 1974.

22. Thomas F. Kavanaugh, "Tabsol - A fundamental Concept for Systems Oriented Languages", Proc. Eastern JCC, 1960, pp. 117-127.

23. Art Lew, "On the Emulation of Flowcharts by Decision Tables", Comm, ACM 25, 12(Nov. 1982), p. 895-905.

24. Peter J.H. King, "The Interpretation of Limited Entry Decision Table Format and Relationships Among the Conditions", Computer J. 12, 4(Nov. 1969), 320-326.

25. Paul Dixon, "Decision Tables and Their Application", Computers and Automation (April 1964) p. 14-19.

26. John C. Cavouras, "On the Conversion of Programs to Decision Tables: Method and Objectives", Comm, ACM 17, 8(Aug. 1984), p. 456-462.

27. M. Ibramsha and V. Rajaraman, "Detection of Logical Errors in Decision Table Programs", Comm, ACM 21, 12(Dec. 1986), p. 1016-1025.

28. Ibramsha and Rajaraman, ibid.

29. P. Piwowarski, "A Nesting Level Complexity Measure", SIGPLAN Notices, Vol. 17, #9, September 1982, p.44.

30. J.J. Tamine, "On the Use of Tree-Like Structures to Symplify Measures of Complexity", SIGPLAN Notices, Vol. 18, #9, September 1983, p.62.

31. G. Cantone, A. Cimitile, L. Sansone, "Complexity in Program Schemes: The Characteristic Polynomial", SIGPLAN Notices, Vol. 18, #3, March 1983, p. 22.

32. R. Wingerter, "To the Editor", SIGPLAN Notices, Vol. 19, #3, March 1984, p. 73.

33. Jaworski and Radhakrishnan, ibid.

34.Herman D. Hughes, "A Programming Language Engineered for Beginners", Comput. Lang. Vol 10, No. 1, pp. 23-36, 1985.

35. J.J. Tamine, ibid.

36.Richard Wingerter, "To the Editor", SIGPLAN Notices Vol 19, #3, March 1984, p. 73.

37. W.M. Jaworski and M. Virard, "Converting a Software Company to a New Technology", Canadian Conference on Industrial Computers, Montreal, Canada, May 1986.

38. Chvalovsky, Vaclav, "Problems with Decision Tables", ACM Forum, Comm. of ACM 19, 12(Dec. 1976), p. 705.

APPENDIX A: TECHNICAL INFORMATION ABOUT THE PROJECT

The JMSS system was implemented on and IBM PC (4.7MHz). This system became the major limitation during the development of the JMSS system due to its very slow processor. However, where needed, other options were explored to improve the system's overall operation. In particular the use of Lotus 123 was considered necessary because the interactive editor in dBase for the implementation development was just too slow.

Lotus 123 templates were used for the initial implementation level development. These templates were designed to match the table format described in section 3.3 (Figure 3.5). Programs were developed using the table approach. Lotus files were then converted to dBase files. This option was selected because Lotus reads the whole file into memory making editing very fast. The initial implementation processes requires a great deal of data entry, and speed is very important.

dBase III Plus software was used for all other features of the system. The author developed all software described in JMSS. dBase III Plus has a number of idiosyncracies that had to be addressed. The biggest problem was the loss of speed due to nested program procedures. The use of a dBase compiler, such as Clipper brought in additional problems due to the lack of 100% compatibility with dBase. Macros, which were used heavily in the application program, were not accepted by the Clipper version available.

The data entry program for the development processes described in section 3.4 was an interactive dBase III application program developed specifically for JMSS. The user selects which process(es) he wants to edit (see Figure 3.2: the processes listed became the main menu with the activities making up the sub menus). The menu driven editor then provides prompts for editing. Information was stored in database format for later review and analysis.

The interpreter and simulator were separate programs developed to test the operation of the software developed. The interpreter actually operated on the information stored in the database. The operation was slow, however, because the logic of the program was also observed (cluster and alternative descriptions were displayed). The slow speed allowed the user to read these descriptions. For small programs this was very useful. The simulator allowed the user to view his program from a logical perspective rather than an operational perspective. The user could select alternatives and the logical descriptions of the cluster and alternative would be displayed.

The program verification feature called ANALYZER allowed the user to review specific cluster sets to test for ambiguous or incomplete steps as described in section 5.3. The algorithm used is based on the use of a Karnaugh map. Mutual exclusion of predicates must be considered when reviewing the analysis of clusters. Perhaps a notation indicating mutual

exclusion should be considered for an improved design.

Program complexity was calculated on some test programs by computer analysis of the programs stored in a database. However, generating a graph of the program produced more important information about complexity than simply calculating a number. The size of the programs, the looping sequences as well as patterns can be viewed from the graphical representation of the control flow of a program. Figure 5.1 is an interesting example of a graphical representation of the control flow of an algorithm.

Numerous report writers were produced. These reports produced tables such as Figure 5.2. The program generator is basically a report of the database information displayed in a specific way. The example given in Figure 5.18 is simply a report of information that is in fact source code.

APPENDIX B: DECISION TABLE FORMAT

Each column represents an alternative in the decision table. Each condition entry may contain zero, 1 or more conditions. If all conditions in the entry are fulfilled the action stub containing zero, 1 or more actions are successively executed. The relationship between successive conditions is the logical "AND": the first condition to be tested is assumed to be preceded by "IF". Example: IF (first condition) AND (second condition),..., AND (last condition). The order in which conditions are listed are of no importance. However, the tables are easier to read if the more important conditions are stated first. Such a sequence may differ from the sequence preferred in programming.

The relationship between successive rules or alternatives is the logical exclusive "OR". The sequence of the alternatives is irrelevant. However, the convention that if an ELSE-rule is used then, to aid readability, it generally appears as the last rule in the table. The satisfied rule is found by determining the particular case in the alternative set that matches the current state. If no alternative is satisfied by the current values then all the actions specified for the ELSE-rule are to be taken in sequence.

By definition, one and only one rule is satisfied for each set of alternatives. The ELSE-rule cannot apply to a case

which is satisfied by one of the alternatives in the set. Any table which includes the ELSE-rule is always complete. The ELSE-rule is, in effect, a default rule used in this environment as detection of an error, or an incomplete list of alternatives within the table. (Note: the ELSE-rule is not shown in the tables, however it can be incorporated into the program generator).

Decision table programming has been studied for many years. Considerable effort has been devoted to algorithms for translating the tables into executable code. Chvalovsky[38] cited the abundance of literature on decision tables describing conversion of decision tables into executable code at the expense of other features of equal if not greater importance. One of the concerns of this thesis is exploring methods for program representation and methods of extracting selected information about programs within a database. Program generation or the conversion of decision tables into executable code is fairly straight forward.

Decision table programming can be approached from two directions. Many beginners start with a completed program and translate the program into decision table form. The second more direct approach, is table programming.

There are difficulties with both approaches. Translating programs to decision tables generally takes longer than the table programming approach. More importantly, the

documentation in the original program will either be lost or be inappropriate when translated to decision tables.

Decision table programming is demanding no matter which approach is used. However, the benefits of using the table structure is considerable. Programs are highly structured and tend to be complete in terms of all possible alternatives defined at a decision point. Computer assisted checking can indicate ambiguous or incomplete rules. This simplifies the checking process.

APPENDIX C: PROJECT SPECIFICATION

The following tables represent templates for project design and specifications. The first four levels were developed using the program "QMAIN". This program provides the templates for the "Identification of Needs", "Requirement Specification", "Design Specifications" and "Software Specifications".

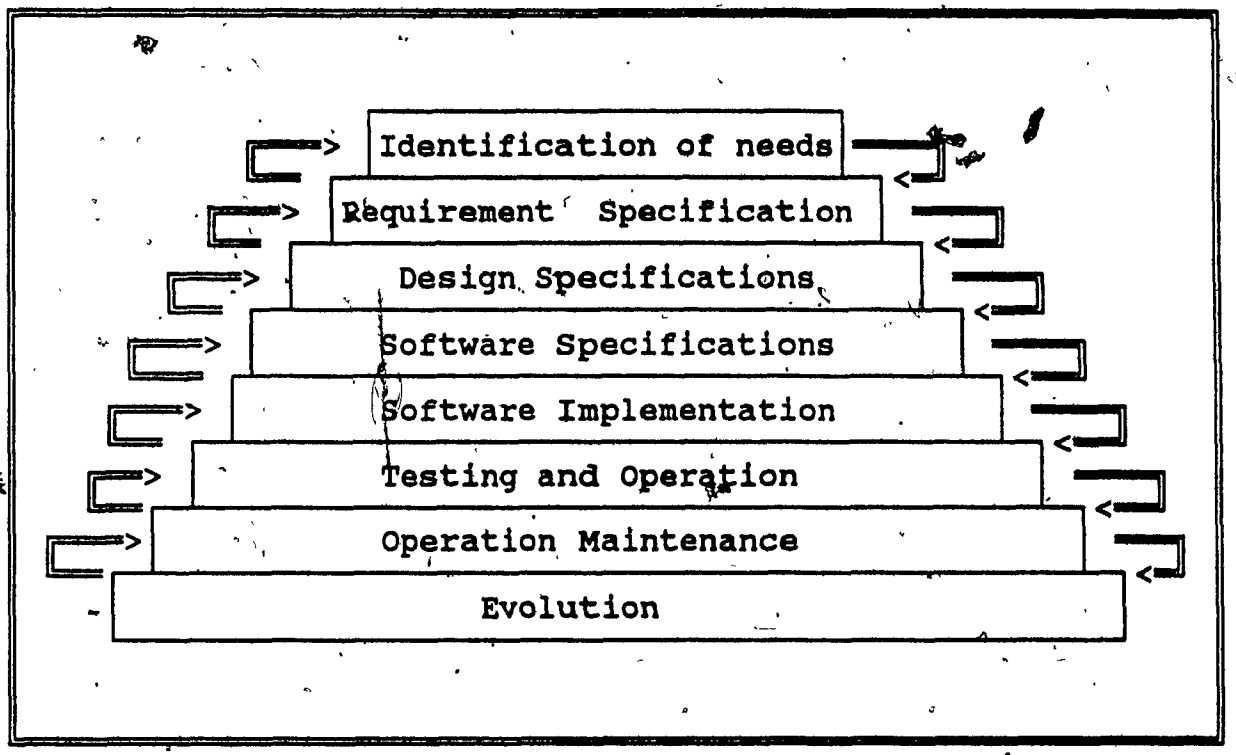


Figure C.1 Stages of System Development

1. Determining System Requirements;
2. Documenting users needs;
3. Partitioning and allocating functional and performance
   requirement to obtain detailed requirements;
4. Analyzing design constraints - putting a limit on design

Figure C.2 Idea Development Features

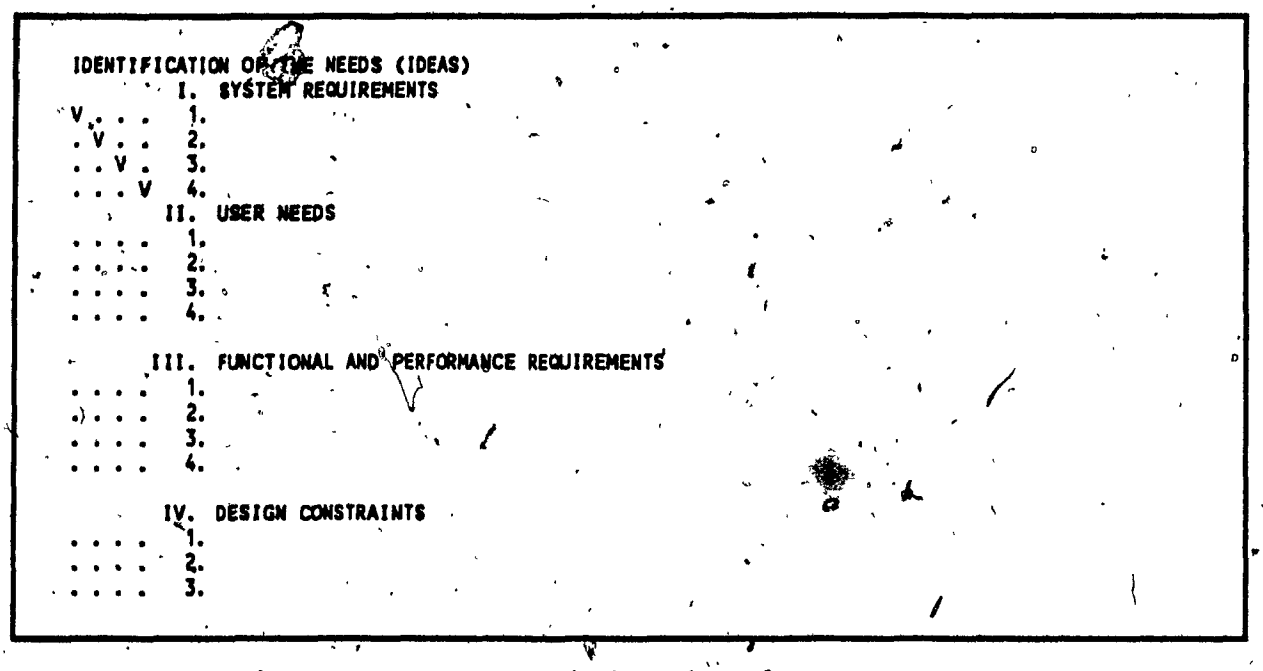


Figure C.3 Identification of the Needs (IDEAS)

1. Functions
2. Performance
3. External interfaces
4. Quality Attributes
5. Design Constraints

Figure C.4 Requirement Specification Features

```
REQUIREMENT SPECIFICATION: SYSTEM REQUIREMENTS

RELATIONSHIPS   SYSTEM REQUIREMENTS
   V . . .   1.
   . V . .   2.
   . . V .   3.
   . . . V   4.
          II. FUNCTIONS
   . . . .   1.
   . . . .   2.
   . . . .   3.
         III. PERFORMANCE
   . . . .   1.
          IV. EXTERNAL INTERFACES
   . . . .   1.
           V. QUALITY ATTRIBUTES
   . . . .   1.
   . . . .   2.
          VI. DESIGN CONSTRAINTS
   . . . .   1.
   . . . .   2.
   . . . .   3.
```

Figure C.5 Requirement Specification

1. Functions
2. Performance
3. External interfaces
4. Internal interfaces
5. Quality Attributes
6. Design Constraints

Figure C.6 Design Specification features

```
DESIGN SPECIFICATION:Function:
 FUNCTIONS
     . . . . 1.
     . . . . 2.
     . . . . 3.
     . . . . 4.
     . . . . 5.
 PERFORMANCE
     . . . . 1.
 EXTERNAL INTERFACE
     . . . . 1.
 INTERNAL INTERFACE
     . . . . 1.
 QUALITY ATTRIBUTES
     . . . . 1.
     . . . . 2.
 DESIGN CONSTRAINTS
     . . . . 1.
```

Figure C.7 Design Specifications: Functions

1. Identification of major features
2. Identification of elements
3. Relationships between major features and elements

Figure C.8 Software Specification Features

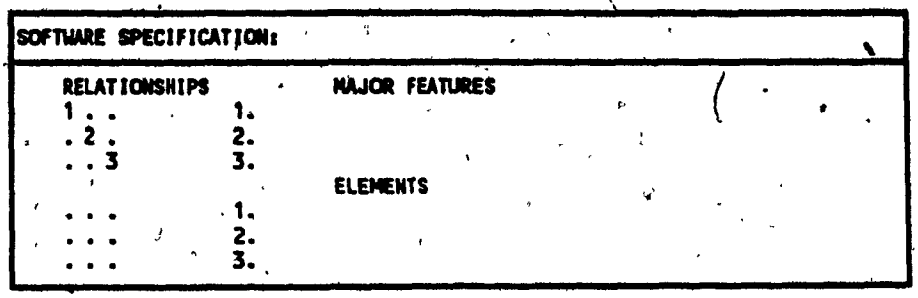


Figure C.9 Software Specification

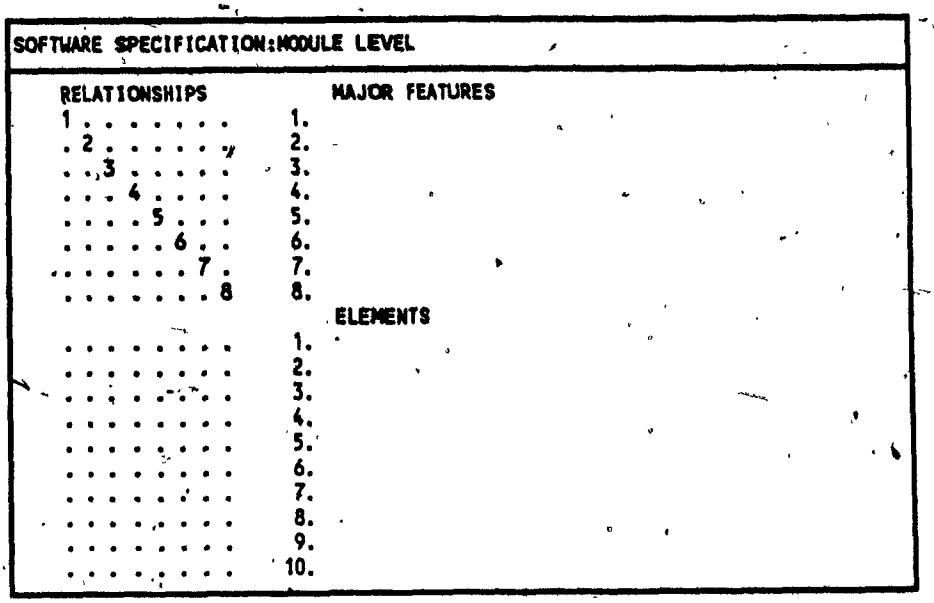


Figure C.10 Software Specification:Module Level

1. Identification of cluster sets
2. Identification of elements of sets (alternatives)
3. Identification of conditions
4. Identification of operations
5. Identification of transactions on cluster sets
6. Identification of objects

Figure C.11 Software Implementation Features

SOFTWARE IMPLEMENTATION: Screen Selection Menu

V . . . . . 1.
. V . . . . . 2.
. . V . . . . 3.
. . . V . . . 4.
. . . . V . . 5.
. . . . . V . 6.
. . . . . V 7.

. . . . . 1.
. . . . . 2.
. . . . . 3.
. . . . . 4.
. . . . . 5.
. . . . . 6.
. . . . . 7.

. . . . . 1.
. . . . . 2.
. . . . . 3.
. . . . . 4.
. . . . . 5.
. . . . . 6.
. . . . . 7.

. . . . . 1.
. . . . . 2.
. . . . . 3.
. . . . . 4.
. . . . . 5.
. . . . . 6.
. . . . . 7.

Figure C.12 Software Implementation

APPENDIX D: PROGRAM REPRESENTATION COMPARISON: CONVENTIONAL CODING, TABLE, GENERATED CODE

The following Figures are examples of the same program solution in different formats: code produced by the program generator, and code produced in the conventional form. Excerpts from these Figures were taken for the Figures in Chapter 5. The Table given in Figure 5.2 is the same problem solution, as is Figure 5.1 showing the graphical representation. However, cluster descriptions given in the conventional coding do not relate to the other cluster codes given in the other representations.

```
CLUSTER=1
EXIT=.F.
DO WHILE .NOT. EXIT
   DO CASE
     *1.0Check each Primary Option selected
     CASE CLUSTER=         1
          DO CASE
               **1.1Current option in range
               CASE tn<=6
                    @ 23,45 SAY "TSTOP = 1 "
                    reblock=.t.
                    toff=1
                    trow=2
                    inTn=.t.
                   CLUSTER=          2
               **1.2All options have been tested:quit
               CASE .not.tn<=6
                   CLUSTER=         25
                   EXIT = .T.
          ENDCASE

     *2.0Check sub options for current current Primary
     CASE CLUSTER=         2
          DO CASE
             **2.1Current sub option to be tested
              CASE inTn
                   @ 23,45 SAY "TSTOP = 2 "
                  CLUSTER=          3
              **2.2All sub options for Primary option have been tested:get next Option
              CASE .not.inTn
                  CLUSTER=          1
          ENDCASE

     *3.0Test if sub Option selected
     CASE CLUSTER=         3
          DO CASE
               **3.1Current sub option has been selected
               CASE &tnow="v" .or. &tnow="V"
                    tnowcont=.t.
                   CLUSTER=          4
               **3.2Current sub option has not been selected; get next sub option
               CASE .not.&tnow="v" .or. &tnow="V"
                    tn=tn+1
                    tnow="t"+ltrim(str(tn))
                   CLUSTER=          2
          ENDCASE

     *4.0Continue testing for selected sub options
     CASE CLUSTER=         4
          DO CASE
               **4.1Test sub options:display current option and sub option
               CASE tnowcont
                    @ 23,45 SAY "TSTOP = 3 "
                    @ trow,2*tspace+7
                   CLUSTER=          5
               **4.2sub option editing complete; get next option
               CASE .not.tnowcont
                    tcontinue=.t.
                    finish=.f.
                    i=0
                    goto top
                    tid=nextid
                   CLUSTER=         10
          ENDCASE
```

```
*5.0Test if Primary Option Selected
CASE CLUSTER=        5
     DO CASE
          **5.1t0: initialize flags
          CASE tnow="t0".and..not. case tnow="t1".and..not.tnow="t2".and..not.tnow="t3"
               .and..not.tnow="t4".and..not.tnow="t5".and..not.tnow="t6"
               tcol=4
               tn=tn+1
               tnow="t"+ltrim(str(tn))
               tnowcont=.t.
               reblock=.t.
              CLUSTER=          8
          **5.2t1:option selected
          CASE .not.tnow="t0".and. case tnow="t1".and..not.tnow="t2".and..not.tnow="t3"
              .and..not.tnow="t4" .and..not.tnow="t5".and..not.tnow="t6"
               @ trow,tcol say "v"
               @ arow,tcol say "v"
               @ 4+tot1,2 say chr(195)+substr(satring,1,2*tspace+1)+chr(180)
               @ trow,2*tspace+7 say "1. IDEA Development Features"
              CLUSTER=          6
          **5.3t2:option selected
          CASE .not.tnow="t0".and..not. case now="t1".and.tnow="t2".and..not.tnow="t3"
               .and..not.tnow="t4".and..not.tnow="t5".and..not.tnow="t6"
               @ 22,40 say'"TNOW='T2'"
              CLUSTER=          8
          **5.4t3:option selected
          CASE .not.tnow="t0".and..not. case now="t1".and..not.tnow="t2".and.tnow="t3"
               .and..not.tnow="t4".and..not.tnow="t5".and..not.tnow="t6"
               @ 22,40 SAY "TNOW='T3'"
              CLUSTER=          8
          **5.5t4:option selected
          CASE .not.tnow="t0".and..not. case now="t1".and..not.tnow="t2".and..not.tnow="t3"
               .and.tnow="t4".and..not.tnow="t5".and..not.tnow="t6"
               @ 22,40 SAY "TNOW='T4'"
              CLUSTER=          8
          **5.6t5:option selected
          CASE .not.tnow="t0".and..not. case now="t1".and..not.tnow="t2".and..not.tnow="t3"
               .and..not.tnow="t4".and.tnow="t5".and..not.tnow="t6"
               @ 22,40 SAY "TNOW='T5'"
              CLUSTER=          8
          **5.7t6:option selected
          CASE .not.tnow="t0".and..not. case tnow="t1".and..not.tnow="t2".and..not.tnow="t3"
               .and..not.tnow="t4".and..not.tnow="t5".and.tnow="t6"
               @ 22,40 SAY "TNOW='T6'"
              CLUSTER=          8
          **5.8None of the tn options selected
          OTHERWISE
               @ 22,40 say "error in 't' value"
              CLUSTER=          8
     ENDCASE
*6.0Are there more sub options to test for?
CASE CLUSTER=        6
     DO CASE
          **6.1
          CASE  if un <= 4
               un=un+1
               toff3=tot1 + 4
               tnowcont=.f.
               unow="u"+ltrim(str(un))
              CLUSTER=          7
          **6.2
          CASE .not. if un <= 4
               tcol=4
               un=un+1
               tnow="t"+ltrim(str(tn))
               tnowcont=.t.
               reblock=.t.
              CLUSTER=          8
     ENDCASE
```

```
    *7.0Test current Sub Option
    CASE CLUSTER=       7
         DO CASE
              **7.1
              CASE  case un=1.and. if u1="v" .or. u1="V".and..not.un=2.and..not.un=3.and..not.un=4
                    @ arow,2*tspace+7 say "1. Determine System Requirements"
                    arow=arow+1
                    tnowcont=.f.
                    set filter to field_type="U1"
                   CLUSTER=         8
              **7.2
              CASE .not. case un=1.and.un=2.and.u2="v" .or. u2="V".and..not.un=3.and..not.un=4
                    @ arow,2*tspace+7 say "2. Document Users Needs"
                    arow=arow+1
                    tnowcont=.f.
                    set filter to field_type="U2"
                   CLUSTER=         8
              **7.3
              CASE .not. case un=1.and..not.un=2.and.un=3.and.u3="v" .or. u3="V".and..not.un=4
                    @ arow,2*tspace+7 say "3. Partition/Allocate functional requirements"
                    arow=arow+1
                    tnowcont=.f.
                    set filter to field_type="U3"
                   CLUSTER=         8
              **7.4
              CASE .not. case un=1.and..not.un=2.and..not.un=3.and.un=4.and.u4="v" .or. u4="V"
                    @ arow,2*tspace+7 say "4. Determine System Requirements"
                    arow=arow+1
                    tnowcont=.f.
                    set filter to field_type="U4"
                   CLUSTER=         8
              **7.5
              OTHERWISE
                   CLUSTER=         8
         ENDCASE
    *8.0Is the cursor off screen?
    CASE CLUSTER=       8
         DO CASE
              **8.1Current Position off screen; reblock screen
              CASE reblock
                   CLUSTER=         9
         ENDCASE
  ENDCASE
ENDDO
```

Figure D.1: Generated Program for "ABC"

CONVENTIONAL CODING

```
****MENU6.PRG
* 1. Initialize: set variables; open files;
* 2. Print items selected from Tn
*    2.1 tn<=6: More items at Tn level to print
*        --->3
*    2.2 tn>6 : No more items at Tn level; end program
*        --->0
* 3. If Tn selected, print else set next Tn
*    3.1 &tnow="v" .or. &tnow="V": Tn has been selected
*        --->4
*    3.2 otherwise: Tn has not been selected; select next Tn
*        --->2
* 4. Select next sub-item for current Tn
*    4.1  Sub-item selected: print list of sub-item descriptions;
*         read first item in list
*         --->5
*    4.2  Sub-item not selected; get next sub-item
*         --->4
*    4.3  Sub-item now out of range: get next Tn
*         --->4
*    4.4  No more items or sub-items remain to be printed
*         --->0
* 5.0 Read input command key and take appropriate action
*    5.1  Move cursor up
*         --->5
*    5.2  Move cursor down
*         --->6
*    5.3  Move cursor to end of this sub-list and start next
*         sub-list
*         --->2
* 6.0 If not at the end of the sub-list, print description
*    6.1 More items in sub-list; display;
*         --->5
*    6.2 No more items in sub-list, insert item into list
*         --->6

*
PUBLIC TOFF,TCOL,TROW,NEXTID,TN,UN,VN,WN
public XN,YN,ZN,TNOWCONT,TNOW,UNOW

tot1=0
tot2=0
tot3=0

if u1='v' .or. u1="V"
     tot1=tot1+1
endif
if u2='v' .or. u2="V"
     tot1=tot1+1
endif
if u3='v' .or. u3="V"
     tot1=tot1+1
endif
if u4='v' .or. u4="V"
     tot1=tot1+1
endif
*count v# fields
if v1='v' .or. v1="V"
     tot2=tot2+1
endif
if v2='v' .or. v2="V"
     tot2=tot2+1
endif
if v3='v' .or. v3="V"
     tot2=tot2+1
endif
*
```

```
 if v4='v' .or. v4="v"
     tot2=tot2+1
 endif
 if v5='v' .or. v5="v"
     tot2=tot2+1
 endif

 if w1="v" .or. w1="v"
     tot3=tot3+1
 endif
 if w2="v" .or. w2="v"
     tot3=tot3+1
 endif
 if w3="v" .or. w3="v"
     tot3=tot3+1
 endif
 if w4="v" .or. w4="v"
     tot3=tot3+1
 endif
 if w5="v" .or. w5="v"
     tot3=tot3+1
 endif
 tot0=tot1
 ****etc*********************************************
  *tspace must be defined
  clear
  SET STATUS OFF
  SELE 1
  use qmast1 index qrec1
 *reindex
  SELE 10
  USE QMAST2 index qrec2
  SET SAFETY OFF
  ZAP
 *** append blank
  SET SAFETY ON
  SELE 1
  sstring=chr(196)+chr(196)+chr(196)+chr(196)+chr(196)
  sstring=sstring+sstring+sstring
  tstring=" . . . . . "
  toff=6
  toff2=1
  tcol=4
  trow=2
  nextid=0
  un=0
  vn=0
  wn=0
  xn=0
  yn=0
  zn=0
  tnow="t0"
  t0="y"
  tn=0
 arow=trow+2
 reblock=.f.
 toff=1
 tstart=1
 tid=nextid
 trecid=0
 trec10=1
 flagstart=.t.
     @ 22,25 SAY "NEXTID TID TNEXTID TOFF TOFF2 TSTART"
     ? "NEXTID TID TNEXTID TOFF TOFF2 TSTART"

 **2:0
 ***MAIN CONTROL LOOP BETWEEN T1,T2,..T6*****************
 do while tn<=6
     @ 23,25 SAY STR(NEXTID,4,0)+"     "+STR(TID,3,0)+STR(TNEXTID,4,0)+"    "+STR(TOFF,3,0)+"  "+STR(TOFF2,4,0)+"  "+S
```

```
    ? STR(NEXTID,4,0)+"     "+STR(TID,3,0)+STR(TNEXTID,4,0)+"   "+STR(TOFF,3,0)+"  "+STR(TOFF2,4,0)+"  "+STR(TSTART,
**2.1
    reblock=.t.
    toff=1
    trow=2
    inTn=.t.
**3.0
    do while inTn
    @ 23,25 SAY STR(NEXTID,4,0)+"     "+STR(TID,3,0)+STR(TNEXTID,4,0)+"   "+STR(TOFF,3,0)+"  "+STR(TOFF2,4,0)+"  "+S
    ? STR(NEXTID,4,0)+"     "+STR(TID,3,0)+STR(TNEXTID,4,0)+"   "+STR(TOFF,3,0)+"  "+STR(TOFF2,4,0)+"  "+STR(TSTART,
      if &tnow="v" .or. &tnow="V"
***3.1
        tnowcont=.t.
**4.0    ****LOOP FOR WITHIN A GIVEN Tn
        do while tnowcont
    @ 23,25 SAY STR(NEXTID,4,0)+"     "+STR(TID,3,0)+STR(TNEXTID,4,0)+"   "+STR(TOFF,3,0)+"  "+STR(TOFF2,4,0)+"  "+S
    ? STR(NEXTID,4,0)+"     "+STR(TID,3,0)+STR(TNEXTID,4,0)+"   "+STR(TOFF,3,0)+"  "+STR(TOFF2,4,0)+"  "+STR(TSTART,
**4.1
            @ trow,2*tspace+7

            do case
            case tnow="t0"
                tcol=4
                tn=tn+1
                tnow="t"+ltrim(str(tn))
                tnowcont=.t.
                reblock=.t.
            case tnow="t1"
                @ trow,tcol say "v"
                @ arow,tcol say "v"
                @ 4+tot1,2 say chr(195)+substr(sstring,1,2*tspace+1)+chr(180)
                @ trow,2*tspace+7 say "1. IDEA Development Features"
                if un <= 4
                    un=un+1
                    toff3=tot1 + 4
                    tnowcont=.f.
                    unow="U"+ltrim(str(un))
                    do case
                    * DETERMINE WHICH "u" *
                        case un=1
                            if u1="v" .or. u1="V"
                                @ arow,2*tspace+7 say "1. Determine System Requirements"
                                arow=arow+1
                                tnowcont=.f.
                                set filter to field_type="U1"
                            endif
                        case un=2
                            if u2="v" .or. u2="V"
                                @ arow,2*tspace+7 say "2. Document Users Needs"
                                arow=arow+1
                                tnowcont=.f.
                                set filter to field_type="U2"
                            endif
                        case un=3
                            if u3="v" .or. u3="V"
                                @ arow,2*tspace+7 say "3. Partition/Allocate functional requirements"
                                arow=arow+1
                                tnowcont=.f.
                                set filter to field_type="U3"
                            endif
                        case un=4
                            if u4="v" .or. u4="V"
                                @ arow,2*tspace+7 say "4. Determine System Requirements"
                                arow=arow+1
                                tnowcont=.f.
```

```
                                    set filter to field_type="U4"
                                endif
                            otherwise
                                ****do nothing; go back and get  next tn
                            endcase
                        else
                         ******GET NEXT Tn
                            tcol=4
                            tn=tn+1
                            tnow="t"+ltrim(str(tn))
                            tnowcont=.t.
                            reblock=.t.
                        endif
                    case tnow="t2"
                        @ 22,40 say "TNOW='T2'"
                        tnowcont=.f.
                    case tnow="t3"
                        @ 22,40 SAY "TNOW='T3'"
                        tnowcont=.f.
                    case tnow="t4"
                        @ 22,40 SAY "TNOW='T4'"
                        tnowcont=.f.
                    case tnow="t5"
                        @ 22,40 SAY "TNOW='T5'"
                        tnowcont=.f.
                    case tnow="t6"
                        @ 22,40 SAY "TNOW='T6'"
                        tnowcont=.f.
                    otherwise
                        @ 22,40 say "error in 't' value"
                    endcase
                    if reblock
                     **********BLOCK SCREEN***************
                            clear
                            tvar=1
                            @ tvar,2 say chr(218)+substr(sstring,1,2*tspace+1)+chr(191)
                            do while tvar<=23
                               @ tvar+1,2 say chr(179)+substr(tstring,1,2*tspace+1)+chr(179)
                               tvar=tvar+1
                            enddo
                            @ 3,2 say chr(195)+substr(sstring,1,2*tspace+1)+chr(180)
                            @ 24,2 say chr(192)+substr(sstring,1,2*tspace+1)+chr(217)
                            toff2=1
                            nextid=0
                            tnextid=1
                            reblock=.f.

                endif
            enddo
            *while tnowcont

            tcontinue=.t.
            finish=.f.
            flagstart=.t.
            goto top
            tid=nextid
            i=24
            toff=tstart
            trow=2
**5.0
            do while .not. finish
        @ 23,25 SAY STR(NEXTID,4,0)+"     "+STR(TID,3,0)+STR(TNEXTID,4,0)+"   "+STR(TOFF,3,0)+"  "+STR(TOFF2,4,0)+"  "+S

        ? STR(NEXTID,4,0)+"     "+STR(TID,3,0)+STR(TNEXTID,4,0)+"   "+STR(TOFF,3,0)+"  "+STR(TOFF2,4,0)+"  "+STR(TSTART,

                do while i=0
                    i=inkey()
                enddo
                do case
**5.1
```

```
case i=5
 *up ->
 *move cursor up
    ?? chr(94)
 if nextid>0
    sele 1
    seek trec
    @ toff+toff3,2*tspace+7 say ltrim(str(nextid))+"."
    @ toff+toff3,2*tspace+10 get desc
    read
    @ toff+toff3,2*tspace+10 say desc
    do case
       case readkey()=15 .or. readkey()=5 .or. readkey()=261  .or. readkey()=271

          *cr*                  *down ->*
          *modify/down ->  *modify/<cr>
          i=24
          toff=toff+1
          nextid=nextid + 1
          skip
          sele 10
          skip
          sele
       case readkey()=3
          i=6
       case readkey()=4 .or. readkey()= 260
          *modify/up ->
          if toff>1
             i=5
             toff=toff-1
             nextid=nextid - 1
             sele 10
             skip -1
             trec=file_rec_n
             sele 1
          else
             i=0
          endif
       otherwise
          i=0
    endcase
  else
    i=0
  endif
**5.2-->6
case i=24
*down ->
*get next item in list
*****PRINTS NEW BLOCK IF SCREEN BLOCK FULL
? "v"
if toff+toff3>23 .or. flagstart
       flagstart=.f.
       nextid=tnextid -1
       tid =nextid + 1
       @ 2,45 say "rec_no="+ltrim(str(rec_no))
       if eof()
          trecid=0
       else
          trecid=rec_no
       endif
       sele 10
       goto bottom
       trec10=rec_no
       sele 1
       toff2=toff
       do while toff+toff3<=23 .and. .not. eof()
          nextid=nextid+1
          @ toff+toff3,2 say chr(179)+substr(tstring,1,2*tspace+1)+chr(179)
          @ toff+toff3,2*tspace+7
          @ toff+toff3,tcol say "v"
```

```
                    @ toff+toff3,2*tspace+10 say desc
                    @ toff+toff3,2*tspace+7 say ltrim(str(nextid))+"."
                    toff=toff+1
                    trec=recno()
                    SELE 10
                    APPEND BLANK
                    REPLACE REC_NO WITH RECNO()
                    REPLACE REC_TYPE WITH "T1"
                    REPLACE FILE_SEL WITH "SELE 1"
                    REPLACE FILE_REC_N WITH TREC
                    SELE 1
                    skip
               enddo
               tnextid=nextid
               do while toff+toff3<=23
                    @toff+toff3,2*tspace+7
                    @toff+toff3,2 say chr(179)+substr(tstring,1,2*tspace+1)+chr(179)
                    toff=toff+1
               enddo
               sele 10
               if trec10>0
                    seek trec10
               else
                    goto top
               endif
               if trecid>0
                    sele 1
                    seek trecid
               else
                    sele 1
                    goto top
               endif
               toff=toff2
               nextid=tid
          endif
          if .not. eof()
**6.1 -->5
               @ toff+toff3,tcol say "v"
               @ toff+toff3,2*tspace+7 say ltrim(str(nextid))+"."
               @ toff+toff3,2*tspace+10 get desc
               READ
               @ 2,40 say "readkey()="+str(readkey())
               @ toff+toff3,2*tspace+10 say desc
               replace rec_no with recno()
               replace id with nextid
***6.2-->5
          else
               tstr="."
               @ toff+toff3,tcol get tstr
               read
               if tstr="v" .or. tstr="V"
                    @ toff+toff3,tcol say "v"
                    append blank
                    replace field_type with unow
                    replace id with nextid
                    replace rec_no with recno()
                    @ toff+toff3,2*tspace+7 say ltrim(str(nextid))+"."
                    @ toff+toff3,2*tspace+10 get desc
                    read
                    TREC=RECNO()
                    SELE 10
                    APPEND BLANK
                    REPLACE REC_NO WITH RECNO()
                    REPLACE REC_TYPE WITH "T1"
                    REPLACE FILE_SEL WITH "SELE 1"
                    REPLACE FILE_REC_N WITH TREC
                    SELE 1
                    @ toff+toff3,2*tspace+10 say desc
                    @ toff+toff3,tcol say LOWER(tstr)
                    tid=tid+1
```

```
                    tnextid=tnextid+1
                else
                  *continue on to next column
                    @ toff+toff3,tcol say "."
                    tcol=tcol+2
                    finish=.t.
                    tstart=toff
                endif
            endif
            do case
                case readkey()=15 .or. readkey()=5 .or. readkey()=261..or.readkey()=271
                    *<cr>                *down->*        *modify/down ->*modify/<cr>
                    i=24
                    toff=toff+1
                    nextid=nextid + 1
                    if .not. eof()
                        sele 10
                        skip
                        sele 1
                        skip
                    endif
                case readkey()=3
                    i=6
                case readkey()=4 .or. readkey()= 260
                    * up"*          *modify/up ->
                    if toff>1
                        i=5
                        toff=toff-1
                        nextid=nextid - 1
                        sele 10
                        skip -1
                        trec=file_rec_n
                        sele 1
                    else
                        i=0
                    endif
                otherwise
                    i=0
            endcase
**5.3-->2
        case i=6
        *<end>
            finish=.t.
            tool=tcol+2
            toff=toff+1
            if toff<toff2
                toff=toff2
            endif
            if nextid<tnextid
                nextid=tnextid
            endif
        otherwise
            i=0
    endcase
enddo
***while .not. finish***
else
    ***if &tnow="V" .or. &tnow = "V"
    tn=tn+1
    tnow="t"+ltrim(str(tn))
endif
enddo
***inTn
enddo
** tn<=6
```

Figure D.2: Conventional Program for "ABC"

APPENDIX E: THE PROCESS MODEL

Formally, a process net related to the cluster set C consists of

1. A table (P,T,F) where P and T are two disjoint sets ($P \cap T = \phi$) called CONDITIONS and OPERATIONS, respectively, and F is a set of states F, each one connecting a condition $p \epsilon P$ to an operation $t \epsilon T$ or vise versa ($F = PxTUTxP$).

2. A set (Q(p)) of states on the cluster set C, associated with the set P(((p) of conditions.

A set (U(t)) of transactions on the cluster set C, associated with the set T(((t) of operations and the corresponding rules.

An operation $t \epsilon T$ is enabled whenever:

a. For each input condition p of t, it corresponds to a result of the query Q(p) that is not empty.

b. There exists at least one set of objects, extracted from the input conditions, for which the rules specified in the transaction U(t) are satisfied.

When the system operates, the actions associated with the operation causes changes to the state and impact on the output conditions of t.